MÁSTER EN LETRAS DIGITALES

TRABAJO DE FIN DE MÁSTER

Curso 2018-2019

Ediciones hiperintertextuales: propuesta de web para navegar entre textos y sus intertextos y herramientas para la automatización de anotaciones TEI

ESPECIALIDAD: Colecciones digitales

APELLIDOS Y NOMBRES: Robles Pérez, María
Saavedra Pascual, Tamara

CONVOCATORIA: Septiembre 2019

CALIFICACIÓN: 9,3

TUTORA: Eva Ullán Hernández
Departamento de Ingeniería del Software e Inteligencia Artificial

AGRADECIMIENTOS

A la poesía, por ser inspiración de los más alocados proyectos.
A Eva, nuestra tutora, por tomar el riesgo de confiar en esta idea, personificando la salvación en sus correos electrónicos.
A David, por ser el guardián de mis desvelos, la voz de mi conciencia y mi aliento en los días más agotadores.
—MARÍA ROBLES PÉREZ

A Eva, por enseñarnos algo nuevo y animarnos con cada correo,
y por haber estado en todo momento al pie del cañón.
A mis padres, por creer en mí y apoyarme en todo lo que hago,
por ser mi mejor soporte y modelo a seguir.
A mi familia y amigos, por todo el humor y amor que aportáis a mi vida
y por inspirarme para dar la mejor versión de mí misma.
A mis abuelos.
—TAMARA SAAVEDRA PASCUAL

ÍNDICE

TABLA DE ILUSTRACIONES

RESUMEN

El siguiente trabajo de fin de Máster consiste en la presentación de un prototipo de edición electrónica para género lírico, centrada en la muestra de intertextos, así como de sus relaciones, a través de una plataforma de lectura hipertextual. Se expone, asimismo, una herramienta de anotación o marcado de textos poéticos automático, siguiendo los estándares de la Text Encoding Initiative (TEI). De igual manera, se plantea, en uno de los epígrafes, la utilización de nuevas etiquetas específicas para el etiquetado de intertextos en el esquema de marcado TEI para XML.

Para la visualización del prototipo, se ha delimitado un breve corpus poético, consistente en una decena de títulos pertenecientes a las *Galerías* de Antonio Machado.

Palabras clave: edición electrónica, intertexto, hipertexto, poesía, automatización, lenguajes de marcado, anotación de textos.

ABSTRACT

The following Master's Thesis consists in the presentation of a prototype of electronic edition for poetry, centered on the sample of intertexts, as well as their relationships, through a hypertextual reading platform. Likewise, it is shown an automatic text annotation tool, which is ruled over the standards of the Text Encoding Initiative (TEI). Similarly, it is proposed a reformulation of the TEI markup standard, including the use of new specific tags for marking up the intertexts.

It has been delimited a brief poetic corpus, consisting of a dozen titles, belonging to Galerías, by Antonio Machado, for the visualization of the prototype.

Keywords: electronic edition, intertext, hypertext, poetry, automation, markup languages, text annotation.

1. INTRODUCCIÓN

El presente trabajo de fin de Máster, centrado en el ámbito de la edición electrónica, comprende el desarrollo de dos objetos distintos: por un lado, la generación de un prototipo de portal web destinado al alojamiento de *ediciones hiperintertextuales*, y por otro, la programación de una herramienta de automatización para el marcado de textos.

Desde mediados de la década de los 60, el fenómeno de la intertextualidad ha sido estudiado con gran interés en el campo de las Humanidades. Este hecho, además, tiene que ver con múltiples y ambiciosos proyectos comparativos a gran escala enmarcados en la disciplina de las Humanidades Digitales que pretenden crear sólidas redes de información basadas en el contraste y análisis de datos, como el proyecto POSTDATA[1] de estandarización y unión de las ediciones electrónicas de poesía, o como el proyecto e-TRAP[2], que pretende comprender, desde un punto de vista diacrónico, la metodología de la reutilización en las obras literarias. En nuestro trabajo, las *ediciones hiperintertextuales* se refieren de forma directa al texto y sus intertextos, y a cómo se lleva a cabo en el interior del portal web la lectura de los mismos, tejiéndose relaciones que apelan a la competencia intertextual del lector.

En este tipo de proyectos, el marcado de textos resulta fundamental, ya que el procesamiento y contraste de los datos, así como su volcado en distintos portales o plataformas web, dependen inevitablemente de las etiquetas que clasifiquen los distintos componentes que conformen dichos textos. Por ello, otra parte a destacar de nuestro trabajo es la muestra de AnoTEI, una herramienta de automatización para el marcado de textos, siguiendo el estándar de codificación de la Text Encoding Initiative (TEI).

En definitiva, el contenido de este trabajo de fin de Máster se relaciona, de forma directa, con las asignaturas cursadas en el Máster en Letras Digitales, habiéndose aplicado, además, en el mismo, conocimientos no impartidos durante el curso académico, lo cual ha resultado en un mayor aprendizaje de la aplicación de conocimientos técnicos al estudio de las Humanidades.

NOTA: Se puede acceder al contenido práctico de nuestro trabajo en el repositorio de GitHub https://github.com/ehi-lldd/ehi-lldd.github.io

[1] http://postdata.linhd.uned.es/
[2] https://www.etrap.eu/

2. ESTADO DE LA CUESTIÓN

Habida cuenta de que presentamos en este proyecto la muestra de dos productos claramente diferenciados, aunque ligados entre sí por el objeto de su desarrollo, debemos plantear, para llevar a cabo la debida contextualización de este trabajo, dos aspectos o marcos principales de estudio: uno de sesgo teórico, relacionado directamente con el estudio de los textos; y otro, aparte, de tipo analítico, unido a la descripción y comparación de ciertos recursos electrónicos del ámbito de la edición y de la anotación de textos.

2.1. EDICIÓN HIPERINTERTEXTUAL: CONCEPTUALIZACIÓN Y MARCO TEÓRICO

A lo largo de las últimas décadas, se han dado muchas y muy variadas definiciones del concepto de *texto*. No vamos a adentrarnos, sin embargo, en llevar a cabo una revisión cronológica o contrastada de las mismas, sino que delimitaremos, más bien, el susodicho concepto, en aras de mostrar nuestro posicionamiento teórico con respecto de esta discusión terminológica. Dicho método nos será útil, además, para indicar cuál sería la idea de *intertexto*, así como de *hipertexto*, a las que nos hemos adherido, pues solo en relación con la palabra *texto* adquieren estas dos últimas absoluta virtualidad crítica.

Siguiendo, en principio, la estela etimológica del término, el *texto* sería, parafraseando a Segre (1985: 368), el «tejido lingüístico de un discurso», esto es, de un producto eminentemente pragmático que se genera en una dimensión comunicativa, tal y como se valora en la gran mayoría de teorías contemporáneas y en disciplinas como el análisis del discurso. Aunque perceptivamente se ha identificado *texto* con *discurso escrito*, y aunque en el proyecto nos hayamos centrado en explorar las posibilidades de esta variante, no podemos desestimar la importancia con que la oralidad ha estado presente en la transmisión de estos y en su definitiva constitución como producto pragmático.

Desde un punto de vista semiótico, además, habría que considerar el concepto de *texto* en una mayor amplitud, en tanto en cuanto se trataría no solo de la materialización de un proceso decodificador de unas normas lingüísticas determinadas, sino que, asimismo, se contemplaría como objeto de una «codificación plural» (Pozuelo, 1988: 71-73; Lotman, 1996: 78). Con esto nos referimos a la relevancia que adquieren los códigos culturales para la consideración de los textos como discurso de plena constitución. Esto

último resulta en un detalle de suma importancia para la interpretación y definición de los textos literarios, en particular. Es en este sentido, en el de la consideración de los textos como el centro de la teorización semiótica, en el que encontraríamos adecuado enunciar aspectos que se suponen necesarios en la conformación de estos, como el de la *coherencia*, que entraña, a su vez, un elemento cohesionador, y que se relaciona directamente con lo que se conoce como *competencia intertextual* o *intertexto lector* (Lozano, Peña-Marín y Abril, 1982: 21-30).

La interpretación de un texto se consideraría el estadio final de su recepción. En este proceso intervienen diversos elementos, como la formación lecto-literaria y el conocimiento del canon o de ciertos *loci* comunes en el mismo. La relación entre ellos, entonces, estaría mediada por la competencia intertextual (Mendoza Fillola, 2001).

Figura 1. Experiencia lectora (Mendoza Fillola, 2001).

Entendemos por *intertextualidad*, entonces, la relación que se establece entre un texto y otros textos, en cuanto que supone una lectura interactiva y crítica de los mismos. El término, como es sabido, alcanzó su madurez crítica en los años 80 con la teoría de Genette, aunque fue acuñado por Julia Kristeva en la década de los 60, como fruto de las tesis dialógicas sobre el discurso de Bajtin. Por ello, esta última considera que un texto es

réplica (función o negación) de otro o de otros textos variados. En la configuración de las redes intertextuales, por lo tanto, resulta fundamental no solo la competencia lectora o la competencia intertextual, sino los paradigmas contextuales contemplados por la semiótica cultural y por la pragmática como determinantes de la empresa de la *reescritura* y de la *relectura* (entendidos estos términos en su vertiente puramente teórica e intertextual, en sentido figurado). Definimos «intertexto», entonces, como el 'objeto de la intertextualidad', siendo su naturaleza textual, con las características que de ello se derivan. El fenómeno de la intertextualidad cuenta con dos tipos de conocimiento sobreentendido por el lector: por un lado, con aquel que se refiere a la producción del texto; y por otro, con aquel que se refiere a la recepción de dicho texto. Ambos saberes terminan imbricados, conformando un conocimiento nuevo acerca de los diálogos e interacciones que los textos, de una forma diacrónica o sincrónica, establecen entre sí (Martínez Fernández, 2001: 38-39).

Estas redes de textos se entretejen en consonancia con dos elementos distintos que afectan a la constitución del discurso: la tipología textual y el empleo de alusiones. Para la definición de las *ediciones hiperintertextuales* hemos tenido en cuenta, en mayor grado, la segunda, puesto que deseábamos simplificar la tarea de la búsqueda de relaciones en base a referencias o alusiones, ya fueran estas explícitas o implícitas en los textos que contuviera la edición. En este punto, entonces, entraría en juego el concepto de la *hipertextualidad*, en sus dos vertientes, tanto literaria, como informática. Con relación a la primera, fue definida por Genette (1989:14) como la relación que se establece entre un texto determinado (*hipertexto*) y otro texto anterior (*hipotexto*), que se transforma o reinterpreta mediante diversas técnicas, como la parodia, el pastiche o la transposición. A pesar de que esta faceta resulta harto interesante, en el proyecto nos hemos enfocado en la vertiente informática, que, acuñada en los 60 por Ted Nelson, hace referencia a la manera en la que se muestran los textos en los dispositivos electrónicos, esto es, a partir de hipervínculos o nodos que los relacionan con otros textos, a los que se puede acceder de forma inmediata. La hipertextualidad es uno de los conceptos base de la conformación de la *World Wide Web*, puesto que la mayoría de las páginas en internet se encuentran codificadas en lenguaje HTML (*Hypertext Markup Language*).

En este sentido, con *edición hiperintertextual* queremos referirnos a una categoría de edición electrónica enfocada en la exposición transparente de las relaciones intertextuales establecidas entre diversos textos y una obra literaria determinada en la que, además, pudiera darse un acceso inmediato, mediante hipervínculos, a los susodichos intertextos.

2.2. CONTEXTUALIZACIÓN: EDICIONES ELECTRÓNICAS Y MARCADO DE TEXTOS

Partiendo de la conceptualización terminológica de las *ediciones hiperintertextuales*, y de la conciencia de que la mayor parte de las ediciones electrónicas publicadas en red se encuentran marcadas bajo los estándares de codificación de la *Text Encoding Initiative* (TEI), exploramos las posibilidades del esquema del Consorcio para la futura anotación del corpus literario a delimitar. La búsqueda de etiquetas, clasificaciones o recursos, en general, para el marcado específico de intertextos dio como resultado el descubrimiento de la carencia de los mismos en el esquema TEI. Hallamos, sin embargo, en este proceso, dos trabajos relevantes para el marco de nuestro proyecto: un artículo sobre codificación intertextual, publicado en el marco del proyecto Orlando de la Universidad de Alberta; y la presentación del *Comic Book Markup Language* (CBML), una iniciativa TEI de la Universidad de Indiana.

En el primero de ellos, y sirviéndonos como principal punto de partida, Susan Brown y su equipo mostraban la posibilidad de declarar una nueva etiqueta de contenido semántico, referida a la intertextualidad (<intertextuality>), en el interior de una DTD (*Document Type Definition*), esto es, como contenido de un archivo descriptor de los elementos, estructura y sintaxis de un documento XML o SGML. En esta DTD (a la cual denominan *Writing Schema*) eminentemente semántica dedicada a aspectos críticos de la escritura de las autoras estudiadas en su proyecto, se hacía mención, además, a aspectos textuales como la producción o la recepción de una obra determinada.

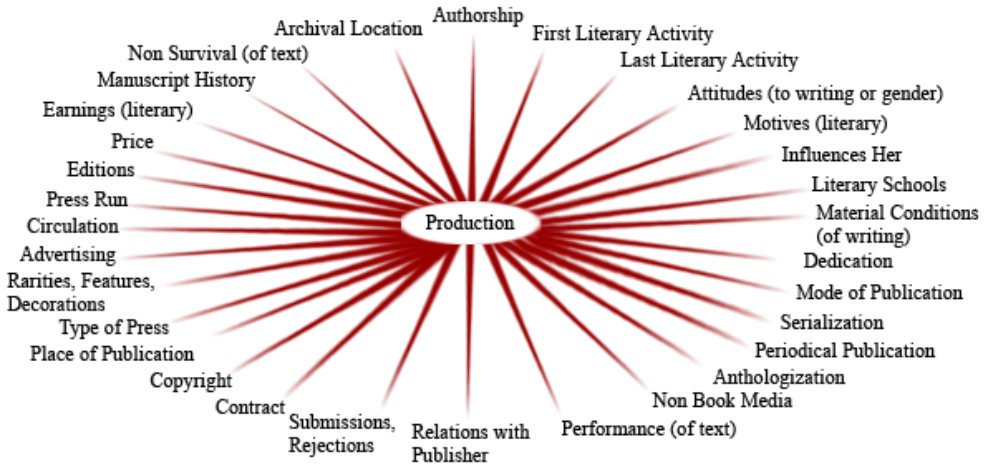

Figura 2. *The Orlando Project Tagsets — Writing Schema*. Etiqueta *Production*.

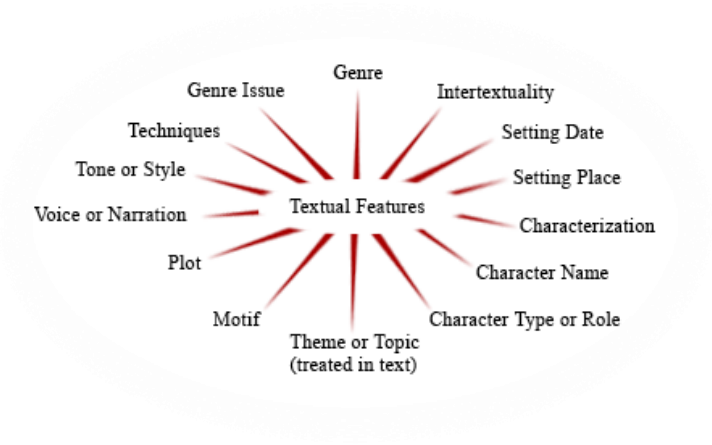

Figura 3. *The Orlando Project Tagsets — Writing Schema*. Etiqueta *Textual Features*.

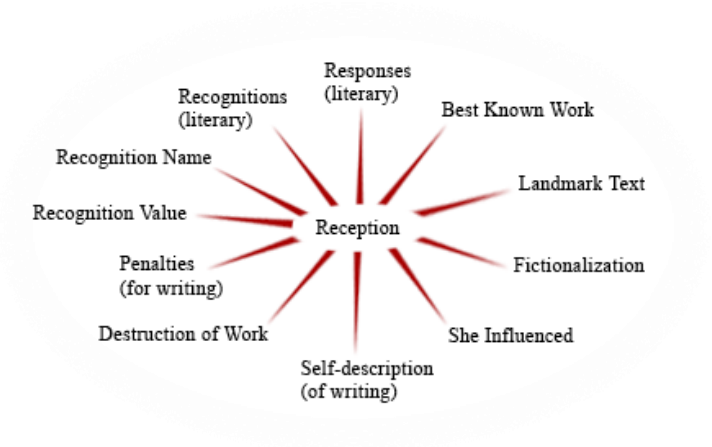

Figura 4. *The Orlando Project Tagsets — Writing Schema*. Etiqueta *Reception*[3].

Esta etiqueta <intertextuality> incluía, para el estudio de la historia de mujeres escritoras británicas, información acerca de sus influencias literarias, de los temas tratados en sus obras, los lugares en los que publicaron, sus relaciones sociales, etc. Los atributos, de hecho, especifican diferentes tipos de relaciones intertextuales: alusión directa, alusión indirecta, cita, parodia, sátira, imitación, adaptación o actualización, precuela, continuación o respuesta, entre otras. A continuación, se muestra un breve ejemplo de su aplicación con marcado SGML:

```
<TINTERTEXTUALITY INTERTEXT="QUOTATION" SEXOFAUTHOR="MALE">
   Helen Dunmore published Burning Bright (whose title comes from Blake's
   lines about the tiger).
</TINTERTEXTUALITY>

<TINTERTEXTUALITY INTERTEXT="QUOTATION" SEXOFAUTHOR="FEMALE">
   Critic Rees-Jones sees in the title of Carol Ann Duffy’s Fifth Last Song:
   twenty-one love poems a reference to Adrienne Rich's “Twenty-One Love
```

[3] Las figuras 2, 3 y 4 han sido tomadas de http://www.artsrn.ualberta.ca/orlando/?page_id=77

```
   Poems” in A Dream of a Common Language, 1978.
</TINTERTEXTUALITY>
```

El hecho de que la anotación de intertextos se llevara a cabo mediante SGML, así como la imposibilidad de acceso a la base de datos del proyecto Orlando, que requiere de una suscripción individual o institucional, nos llevó a descartar la implementación directa de su modelo en nuestro prototipo. Sin embargo, esto nos condujo a encontrar, en acceso abierto, el esquema del proyecto CBML de marcado de cómics a través de los estándares TEI. En este, se introducen tres elementos personalizados u originales: <cbml:panel>, <cbml:balloon> y <cbml:caption>, correspondientes a la anotación de viñetas, de bocadillos y leyendas o paneles de narración, respectivamente. Por cuestiones de género literario, no contemplamos la posibilidad de reutilizar sus etiquetas, pero nos resultó de inspiración para comprobar que, efectivamente, el esquema de TEI podía ser modificado a la sazón de las necesidades propias de cada texto o elemento a codificar semánticamente.

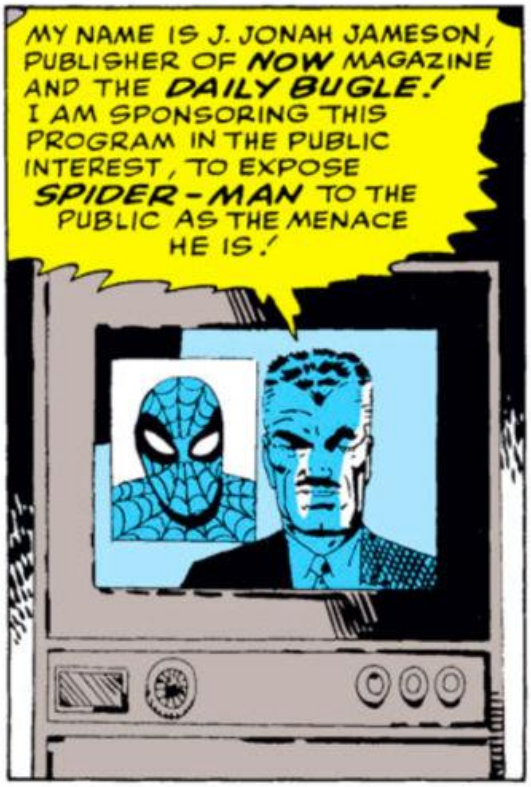

Figura 5. Viñeta de *The Amazing Spider-Man* y su correspondiente etiquetado con CBML (Walsh, 2012)

```
<cbml:panel characters="#spidey #jjj" n="3" xml:id="eg_ae1">
   <cbml:balloon rendition="#uc" type="audio" subtype="telecast" who="#jjj"
   xml:id="eg_006">
      My name is J. Jonah Jameson, publisher of <title rendition="#b">
      Now</title> magazine and the <title rendition="#b">Daily Bugle</title>
      <emph rendition="#b">!</emph> I am sponsoring this program in the
      public interest, to expose <emph rendition="#b">Spider-Man</emph> to
      the pubic as the menace he is!
   </cbml:balloon>
</cbml:panel>
```

En este contexto, surgieron las dos principales motivaciones de este trabajo de fin de Máster: la generación de un prototipo o muestra de edición hiperintertextual y el desarrollo de una propuesta de etiquetado específico y sencillo para intertextos, siguiendo la codificación establecida por los estándares de la TEI.

3. OBJETIVOS GENERALES Y MOTIVACIONES

En el marco de debate suscitado por los contenidos de las diversas asignaturas del Máster en Letras Digitales, en relación al ámbito de la edición electrónica, así como de la anotación de textos, y teniendo en cuenta el contexto explicado previamente, surgió la principal motivación que suscitaría la consecución de este proyecto técnico, traducido a trabajo de fin de Máster: deseábamos presentar un prototipo o muestra digital que supusiera la apertura o exploración de una nueva vía de publicación en red para el género lírico o poético.

La hipótesis e idea original del prototipo a desarrollar, correspondiente a lo que denominamos «edición hiperintertextual», responde a los siguientes objetivos generales:

1) generar un prototipo de interfaz atractiva y amigable para alojar ediciones intertextuales de cualquier obra literaria;
2) reinterpretar el concepto de intertextualidad en la era posdigital mediante el hipertexto y las relaciones hipertextuales;
3) adherir la muestra a la filosofía del acceso abierto y accesible, revistiéndola de un carácter divulgativo y educativo a todos los niveles, con licencia GPL[4];
4) aportar un nuevo recurso al catálogo estándar de tipos de ediciones electrónicas, siguiendo las directrices de la Text Encoding Initiative (TEI);
5) y apostar por la simplificación de los procesos de marcado de textos mediante la máxima automatización posible de los mismos.

De todas las motivaciones anteriores, se derivan ciertas metas específicas, entre las cuales se encontrarían las que se enuncian a continuación:

1) Desarrollo e implementación de una muestra prototípica de un futuro portal web donde alojar el modelo de edición hiperintertextual, haciendo uso del CMS[5] WordPress, por un lado, y de HTML, por el otro, obteniéndose, así, dos portales idénticos, aunque en distintos soportes.
2) Teorización sobre el esquema de marcado XML-TEI en relación a la noción de «intertexto» y discusión ejemplificada acerca de la inclusión de una nueva etiqueta, cuyo contenido se corresponda con esta información textual.

4 https://www.gnu.org/licenses/gpl-3.0.html

5 *Content Management System* (‘sistema gestor de contenidos’): permite añadir, modificar o eliminar todo lo relacionado con los datos y el diseño de un sitio web.

3) En consonancia con la filosofía del proyecto, y con las funcionalidades ofrecidas por WordPress, en caso de que fuera posible, se procedería a la instalación de ciertos *plugins* que contribuyeran a la constitución de un portal web colaborativo. Para ello, se añadiría, como mínimo, un foro de discusión, así como alguna herramienta de anotación.
4) Integración de funcionalidades de marcado métrico en el portal web, como información métrica del poema y un pintador de rimas, mediante botones interactivos.
5) Integración de *tags*, o contenido informativo sobre las áreas temáticas tratadas en los versos que se muestren en la vista del portal.
6) Integración de accesos directos bajo cada poema a los diversos intertextos del mismo.
7) Desarrollo de una aplicación o programa con el que se procure automatizar la transformación de documentos .txt a archivos .xml, marcados siguiendo los estándares de la TEI.

4. METODOLOGÍA

Para la consecución de los objetivos que fundamentan nuestro proyecto hemos empleado diversas técnicas y lenguajes de programación, como el marcado en XML o el desarrollo en Python. La adecuada comunicación entre los mismos ha procurado, finalmente, la implementación del prototipo en un portal web mediante maquetado HTML, en consonancia con el *framework* de Twitter, Boostrap, por un lado, y, por otro, haciendo uso del CMS WordPress.

En la planificación del proyecto, contemplamos las siguientes fases:

a) En primer lugar, detallamos los objetivos del proyecto y las metas a alcanzar en la muestra prototípica de edición web. En esta fase, además, llevamos a cabo un proceso de investigación bibliográfica acerca del concepto de intertexto y de las técnicas digitales aplicadas a la generación de ediciones electrónicas. De igual manera, analizamos otros productos en red, como el proyecto Orlando o una propuesta de lenguaje de marcado para cómics, con miras a describir el marco en el que se insertaría nuestro trabajo (§ESTADO DE LA CUESTIÓN).

b) Seguidamente, delimitamos un breve corpus de poemas que pudiera servirnos como muestra de contenido para nuestro prototipo de edición hiperintertextual.

c) A continuación, llevamos a cabo una fase técnica de desarrollo, en la cual contemplamos la generación de un programa de marcado automático en XML, siguiendo los estándares de TEI.

d) En la penúltilma fase, realizamos cierta labor de marcado manual en XML, completando las hojas generadas automáticamente en el estadio previo del proyecto. En este punto, además, intentamos integrar el nuevo etiquetado de intertextos en nuestro documento XML. Generamos, al final de todo este proceso, una hoja de transformaciones XSLT que nos permitiera simplificar el proceso de darle un estilo uniforme a todos los poemas.

e) Por último, alcanzamos el estadio de implementación, en el cual diseñamos la interfaz de usuario para el portal web, donde alojaríamos una muestra de nuestro prototipo, aplicándole diversas funcionalidades. Desarrollamos esta fase de dos maneras:

i. mediante el despliegue de un servidor local, volcando todo el contenido de la edición en el CMS WordPress.

ii. mediante maquetado HTML, en local, con ayuda del *framework* de Twitter, Bootstrap.

4.1. DELIMITACIÓN DEL CORPUS DE MUESTRA: MACHADO HIPERINTERTEXTUAL

En una segunda fase del proyecto, habiendo trazado el marco teórico y contextual en el que se insertaría nuestro prototipo, nos centramos en delimitar el corpus que sirviera como muestra de contenido para nuestra propuesta de *edición hiperintertextual*.

Escogimos para nuestro propósito diez de los poemas de *Galerías*, incluidos en *Soledades. Galerías. Otros poemas* (1907) de Antonio Machado: el I, el II, el VI, el IX, el X, el XVII, el XIX, el XXIV, el XXX y el XXXI. Para la obtención de los textos acudimos al portal de acceso abierto y gratuito Wikisource[6], así como a la edición en papel, dirigida por Manuel Alvar y publicada en Austral en 2010. El proceso de selección se sometió a una lectura previa del epígrafe completo de la obra machadiana, con una posterior criba de piezas en base a las áreas temáticas y/o semánticas tratadas en las mismas. Generamos, así, un documento en Excel donde se llevaba a cabo un recuento de las concomitancias y apariciones de ciertos *loci* comunes en los poemas de *Galerías*; de esta manera, contamos con el tratamiento de conceptos como el del «alma», los «sueños», el «viento» o la «tarde», entre otros.

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV	XV
alma	4		1	1	1					1	1			1	
sueño(s)	1	1	2	2	2	2			1	1		1			
viento	1	1						2							1
tarde						1				1			1		
niño(a/os/as)						3									
flor	1	1					1	1			1				
jardín(es)						1		1		1					
árbol(eda)															
primavera										1					
oro										1					
sol	2	2				1									
sombra / sombrío													1		
galería	1			1						1					
agua		2					2								
dolor(es)	1								1						
Total	**7**	**5**	**2**	**3**	**2**	**5**	**2**	**3**	**2**	**7**	**2**	**1**	**2**	**1**	**1**

Figura 6. Tabla de concomitancias temáticas (1)

[6] https://es.wikisource.org/wiki/Galer%C3%ADas

	XVI	XVII	XVIII	XIX	XX	XXI	XXII	XXIII	XXIV	XXV	XXVI	XXVII	XXVIII	XXIX	XXX	XXXI
alma		1	2	1				2					4			
sueño(s)		1	1				1		3			1				
viento	1		1	1		1						1			2	3
tarde	2	1			1	1									1	1
niño(a/os/as)		2					1					1				
flor									3	2	1					
jardín(es)							1		1							
árbol(eda)				1	1					1					1	1
primavera										1						
oro				1	1				1							1
sol				1					1							1
sombra / sombrío	1	1	1		1	1	1								1	1
galería																
agua											2	1			1	
dolor(es)		1									3					
Total	**3**	**6**	**4**	**5**	**4**	**3**	**4**	**1**	**5**	**3**	**3**	**4**	**1**	**0**	**5**	**6**

Figura 7. Tabla de concomitancias temáticas (2)

Para cada uno de los poemas escogidos, se tomaron, a su vez, uno o dos recursos literarios intertextuales, pertenecientes tanto a obras de otros autores, como a piezas contenidas en el resto de la producción machadiana. Por ejemplo, en el caso de la novena *galería*, consideramos apropiado adjuntar como intertexto un poema de Félix Grande, titulado «Como una inundación».

La relación de obras, con sus respectivos intertextos, sería la que sigue:

Obra (*Galerías*)	Intertexto(s)		
	Autor/es	Título	Recurso
I. Introducción	Antonio Machado	En nuestras almas todo… (XXVI)	*Soledades. Galerías y otros poemas* (1907). Ed: Austral (2010)
II. Desgarrada la nube...	Juan Ramón Jiménez	Nubes (XXXV)	*Sonetos Espirituales (1914-15)*. Ed.: Facediciones (2012)
VI. ¡Y esos niños en hilera…!	Antonio Machado	Los cantos de los niños (VIII)	*Soledades. Galerías y otros poemas* (1907). Ed: Austral (2010)
IX. Hoy buscarás en vano	Félix Grande	Como una inundación	*Biografía: poesía completa (1958-1984)*. Ed: Anthropos (1989)
X. Y nada importa...	Gabriel Celaya	La vida, ahí fuera	*Poemas órficos* (1978)

XVII. Es una tarde cenicienta y mustia...	Eladio Cabañero	Ocaso	*Palabra compartida (Antología poética)*. Ed: Biblioteca de Autores Manchegos. C. Real (2014)
XIX. Desnuda está la tierra	Eladio Cabañero	Ocaso	*Palabra compartida (Antología poética)*. Ed: Biblioteca de Autores Manchegos. C. Real (2014)
	Juan Ramón Jiménez	Andando (sueño)	*Antología de textos juanrramonianos*. Ed: Biblioteca Virtual Cervantes
XXIV. El rojo sol de un sueño...	Antonio Machado	Caminante, son tus huellas… (XXIX)	*Proverbios y cantares. Poesías completas*. Ed: Austral (2010)
XXX. Los árboles conservan...	Federico García Lorca	Sueño	*Libro de poemas (1919). Obras completas*. Ed: Aguilar (1980)

Figura 8. Relación de poemas de *Galerías* y sus intertextos

4.2. ANOTEI: AUTOMATIZACIÓN DEL MARCADO CON PYTHON

Una vez realizada la selección de los poemas que deseábamos incluir en nuestra edición, decidimos buscar una herramienta que automatizase al máximo posible la anotación de estos en un archivo XML ajustado al estándar de TEI. La primera que utilizamos fue OxGarage[7], proporcionada por el propio Consorcio de TEI, quienes la definen como «a web, and RESTful, service to manage the transformation of documents between a variety of formats» (2011). OxGarage permite la conversión de documentos de diversas extensiones, bien sean archivos de texto (.docx, .odt, .txt, por citar algunos de los más extendidos), presentaciones (.pptx, .odp) u hojas de cálculo (.csv, .xslx o .tsv) a una selección de formatos que varía dependiendo del tipo de documento que queramos convertir. Así, si seleccionamos un archivo de texto plano (.txt), OxGarage nos ofrecerá convertirlo a CSV, ePub, LaTeX, Word, PDF, TEI P5 XML, TEI Simple XML o XML, entre otros. Para probarlo, emplearemos un archivo de texto plano (.txt) que contiene el poema «Hoy buscarás en vano», una de las *Galerías* de Antonio Machado (2010):

[7] https://oxgarage2.tei-c.org/

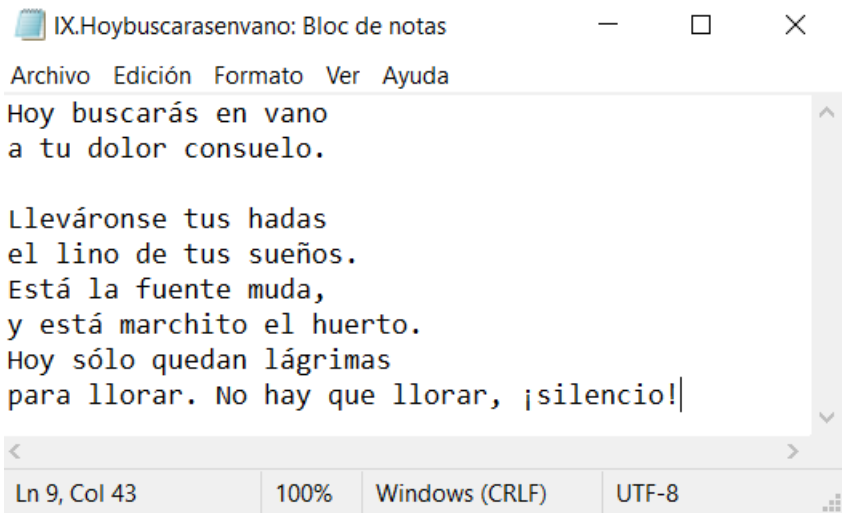

Figura 9. Poema «Hoy buscarás en vano» en .txt

Seleccionamos la opción «Convert to: TEI P5 XML Document». El resultado es el siguiente:

```
<TEI xmlns="http://www.tei-
c.org/ns/1.0"><teiHeader><fileDesc><titleStmt><title>Untitled
Document</title><author/></titleStmt><editionStmt><edition><date/></edition><
/editionStmt><publicationStmt><p>no publication statement
available</p></publicationStmt><sourceDesc><p>Written by
OpenOffice</p></sourceDesc></fileDesc><revisionDesc><listChange><change><name
/><date/></change></listChange></revisionDesc></teiHeader><text><body><p>Hoy
buscarás en vano </p><p>a tu dolor consuelo. </p><p>Lleváronse tus hadas
</p><p>el lino de tus sueños. </p><p>Está la fuente muda, </p><p>y está
marchito el huerto. </p><p>Hoy sólo quedan lágrimas </p><p>para llorar. No
hay que llorar, ¡silencio!</p></body></text></TEI>
```

Pese a tratarse de una herramienta oficial que en primera instancia prometía ser bastante útil, apreciamos que presenta algunas limitaciones que hacen que el documento descargado no nos sea válido. La más importante consiste en que la web no parece tener en cuenta todos los módulos determinados por TEI, ya que en ningún momento nos ofrece la posibilidad de elegir, por ejemplo, si queremos nuestro texto anotado como verso o como prosa.

Entre las directrices para la codificación ofrecidas por el Consorcio, aparecen definidos una serie de módulos, según la naturaleza del texto que va a ser anotado, como *corpus*, *dictionaries*, *drama*, *spoken* o *verse* (Text Encoding Initiative Consortium, 2019)[8]. Cada uno de estos módulos cuenta con sus propios elementos y atributos para la

[8] Para todas las referencias a esta obra, remitimos a la página del Consorcio donde aparecen las directrices en diferentes formatos. Las definiciones en español han sido extraídas de la versión web en dicho idioma, que se encuentra en https://www.tei-c.org/release/doc/tei-p5-doc/es/html/index.html

anotación, algunos de los cuales han sido diseñados *ad hoc* para un único módulo, mientras otros son comunes a todos los demás y forman parte del módulo *core*.

Nuestro texto de ejemplo es un poema, por lo que deberíamos emplear en la anotación los elementos propios del módulo *verse*, como son <lg> —*line group* o 'grupo de versos', «contiene un grupo de versos que funcionan como una unidad formal, p.ej. una estrofa, un refrán, un estribillo, etc.» (Text Encoding Initiative Consortium, 2019)— y <l> —*line* o 'verso', «contiene un único verso, posiblemente incompleto» (Text Encoding Initiative Consortium, 2019)—. Sin embargo, del documento descargado se interpretaría que el texto está en prosa, puesto que el elemento <p>, *paragraph* o 'párrafo', «marca párrafos en prosa» (Text Encoding Initiative Consortium, 2019).

Por otro lado, toda la anotación aparece en una sola línea, con los elementos puestos uno detrás de otro y sin aplicar el sangrado o indentación que suelen presentar documentos escritos con lenguajes de marcado (HTML, XML). Este sangrado no es obligatorio, pero su uso está ampliamente extendido ya que facilita no solo la lectura del documento, sino también la localización de los elementos y la identificación de las relaciones de jerarquía y parentesco que estos tienen entre sí. Por ello, aunque existen multitud de herramientas, tanto en la web como en funciones de los principales editores de código, que tabulan los documentos de manera automática, resulta inusual que OxGarage no lo aplique directamente en su exportación. Así, si aplicamos Indent XML[9] en Sublime Text 3 al documento anterior, quedaría de la siguiente manera:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
   <teiHeader>
      <fileDesc>
         <titleStmt>
            <title>Untitled Document</title>
            <author/>
         </titleStmt>
         <editionStmt>
            <edition>
               <date/>
            </edition>
         </editionStmt>
         <publicationStmt>
            <p>no publication statement available</p>
         </publicationStmt>
         <sourceDesc>
            <p>Written by OpenOffice</p>
```

[9] https://packagecontrol.io/packages/Indent%20XML

```
        </sourceDesc>
      </fileDesc>
      <revisionDesc>
         <listChange>
            <change>
               <name/>
               <date/>
            </change>
         </listChange>
      </revisionDesc>
   </teiHeader>
   <text>
      <body>
         <p>Hoy buscarás en vano </p>
         <p>a tu dolor consuelo. </p>
         <p>Lleváronse tus hadas </p>
         <p>el lino de tus sueños. </p>
         <p>Está la fuente muda, </p>
         <p>y está marchito el huerto. </p>
         <p>Hoy sólo quedan lágrimas </p>
         <p>para llorar. No hay que llorar, ¡silencio!</p>
      </body>
   </text>
</TEI>
```

Tras realizar una búsqueda y estudiar otras herramientas que sirviesen para facilitar la tarea de realizar anotaciones en TEI, como WebAnno, jEdit, Serna Free, Pandoc o Wed[10], ninguna ofrecía un resultado cercano a lo que nosotras buscábamos, de modo que decidimos implementar un programa sencillo que automatizase el proceso de anotación de poemas con TEI. Para ello, utilizamos el lenguaje de programación Python, y decidimos desarrollarlo en Google Colaboratory, «un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube» (2017). Está, además, vinculado con Google Drive, por lo que varias personas pueden tener acceso al archivo con el código en su última versión en todo momento, y permite descargarlo tanto en formato Jupyter Notebook (.ipynb) como Python (.py). Por último, es posible importar y exportar documentos, como el archivo de texto del que partíamos, de una manera sencilla tanto desde Google Drive como desde nuestro directorio local.

Nuestro objetivo era, a través de la importación de un poema en .txt como el que veíamos en la FIGURA 9, generar y descargar un archivo .xml que incluyese la cabecera

[10] https://webanno.github.io/webanno/; http://www.jedit.org/; https://github.com/ydirson/serna-free; https://pandoc.org/; https://github.com/mangalam-research/wed

de TEI y realizase automáticamente la anotación de los elementos <l> para cada línea y <lg> para cada estrofa, cada uno de ellos con su correspondiente identificador único, además de un <lg> que marcase todo el poema y un elemento <head> en el que se registrase el título del mismo, solventando así las dificultades que localizamos en OxGarage. Como ejemplo, el resultado esperado para el mismo poema de la FIGURA 9 sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
   <teiHeader>
      <fileDesc>
         <titleStmt>
            <title></title>
         </titleStmt>
         <publicationStmt>
            <p></p>
         </publicationStmt>
         <sourceDesc>
            <p></p>
         </sourceDesc>
      </fileDesc>
   </teiHeader>
   <text>
      <body>
         <head>Hoy buscarás en vano</head>
         <lg xml:id="P0009">
            <lg xml:id="P0009E0001">
               <l xml:id="P0009V0001">Hoy buscarás en vano</l>
               <l xml:id="P0009V0002">a tu dolor consuelo.</l>
            </lg>
            <lg xml:id="P0009E0002">
               <l xml:id="P0009V0003">Lleváronse tus hadas</l>
               <l xml:id="P0009V0004">el lino de tus sueños.</l>
               <l xml:id="P0009V0005">Está la fuente muda,</l>
               <l xml:id="P0009V0006">y está marchito el huerto.</l>
               <l xml:id="P0009V0007">Hoy sólo quedan lágrimas</l>
               <l xml:id="P0009V0008">para llorar. No hay que llorar,
¡silencio!</l>
            </lg>
         </lg>
      </body>
   </text>
</TEI>
```

Explicamos a continuación el código .py paso por paso[11], mostrando imágenes con fragmentos del código seguidas de su correspondiente descripción:

```
# El usuario indica el poema
print("Indica qué poema que quieres transformar")
archivoPoema = input("Nombre del archivo (sin extensión: ")

# Abre el archivo indicado, añadiendo la extensión .txt y codificación
# UTF-8. Lee el poema línea a línea y lo guarda en una variable
with open(archivoPoema + '.txt', mode='r', encoding="utf-8") as poema:
  lineasPoema = poema.readlines()
```

En primer lugar, se solicita al usuario el nombre del archivo que quiere transformar sin extensión (esto es, sin añadir .txt al final, ya que el programa lo hace automáticamente), el cual deberá localizarse en la carpeta a la que accede Python. Se abre el archivo añadiendo la extensión e indicando su codificación, leyendo el poema línea por línea y almacenándolo en la variable «lineasPoema».

```
# Si el archivo está vacío, lo indica antes de continuar. Si no, pide
# al usuario que introduzca el título del poema que aparecerá anotado
# y que añada un identificador
if lineasPoema == []:
  print("El archivo que has seleccionado está vacío.")
else:
  tituloPoema = input("Título del poema: ")
  idPoema = input("ID del poema (se recomienda introducir 4 dígitos): ")
```

Se evalúa si el documento está en blanco, en cuyo caso se indica con un texto para que el usuario lo revise. En caso contrario, el programa continúa, creando dos variables que le pedirán introducir el título del poema y el identificador que quiera asignarle. Como veremos a continuación, las estrofas y los versos llevarán asociados asimismo sus propios identificadores únicos, con los formatos Exxxx y Vxxxx, respectivamente —donde «x» es un dígito entre el 0-9 y la numeración es sucesiva, comenzando en 0001—, y por ello se recomienda que el identificador introducido sea un número de cuatro dígitos, acorde con el resto de anotaciones; no obstante, el usuario puede introducir cualquier tipo de identificador, incluso uno no numérico.

```
  # Cabecera de TEI
  header = """<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
   <teiHeader>
      <fileDesc>
```

[11] Explicamos aquí el grueso del código tal como funcionaría en cualquier consola de Python o cuaderno, y posteriormente indicaremos los cambios que añadimos al pasarlo a Google Colaboratory, aprovechando las ventajas de su librería «import».

```
         <titleStmt>
            <title></title>
         </titleStmt>
         <publicationStmt>
            <p></p>
         </publicationStmt>
         <sourceDesc>
            <p></p>
         </sourceDesc>
      </fileDesc>
   </teiHeader>
   <text>
      <body>\n"""
```

Seguidamente, escribimos en la variable «header» una cadena de texto que contiene los elementos de TEI que abrirán todos nuestros documentos; esta parte es obligatoria e invariable en todos ellos. Hemos decidido incluir las tabulaciones pertinentes mediante espacios[12] para que el documento que creemos salga ya adecuadamente sangrado, solventando así una de las carencias que detectábamos en OxGarage. Finalizamos el bloque de texto con un salto de línea «\n» para que al encadenar este texto con los siguientes que definamos, estos no aparezcan a continuación en la última línea, sino que empiecen en una nueva.

```
  # Incluye el título del poema introducido por el usuario
  head = "         <head>{}</head>\n".format(tituloPoema)

  # Incluye el ID del poema introducido por el usuario
  body = '         <lg xml:id="P{}">\n'.format(idPoema)

  # Cierre de TEI
  footer = """            </lg>
         </lg>
      </body>
   </text>
</TEI>"""
```

A continuación, recuperamos los datos que habían sido facilitados por el usuario y los introducimos en sus posiciones correspondientes dentro del documento TEI: el título en la etiqueta <head> y el identificador en el primer <lg>, que engloba todo el poema.

[12] En un principio, representamos cada tabulación con el símbolo reservado «\t», pero el espacio que marca es mucho mayor y si se añaden anotaciones posteriormente la visualización puede verse afectada, por lo que sería contraproducente. Optamos, en su lugar, por la indentación con tres espacios, igualmente estética y más funcional.

También definimos, al igual que hicimos con la cabecera, las etiquetas de cierre que se incluirán al final del documento para que este conforme un XML válido.

```
  # Variable que se irá reescribiendo a lo largo del poema para ir
  # añadiendo los versos según se vayan leyendo. Contiene, además,
  # la anotación de la primera estrofa
  salida = '            <lg xml:id="P{}E0001">\n'.format(idPoema)

  # Contadores para la numeración de versos y estrofas
  nverso = 0
  nestrofa = 1
```

La variable «salida» contiene la línea que define la primera estrofa del poema y su identificador[13]. Reutilizaremos esta variable a lo largo del código, pues iremos iterando sobre ella y reescribiéndola constantemente, añadiendo las líneas de nuestro documento. Definimos, asimismo, el comienzo de nuestros contadores, a los que incrementaremos un número de manera progresiva para los identificadores de los versos y las estrofas. Recordamos que la primera estrofa la habíamos introducido manualmente, por lo que este contador tendrá que empezar en el número uno del índice para que la siguiente sea numerada como «E0002».

```
  # Recorre todas las líneas menos la última
  for linea in lineasPoema[:-1]:
    # Si la línea solo tiene un carácter, este es un salto de línea
    # Cierra la estrofa y abre la siguiente
    if len(linea) == 1:
      nestrofa += 1
      cestrofa = '{:04.0f}'.format(nestrofa)
      salida = salida + '            </lg>\n            <lg
xml:id="P{}E{}">\n'.format(idPoema, cestrofa)
    # Todas las líneas de verso salvo la última acaban con un salto de
    # línea, ignora el último carácter para que l se cierre en la misma
    # línea
    else:
      linea = linea[:-1]
      nverso += 1
      cverso = '{:04.0f}'.format(nverso)
      salida = salida + '                <l
xml:id="P{}V{}">{}</l>\n'.format(idPoema, cverso, linea)

  # si la línea es la última, no hace falta que ignore el último carácter
  linea = lineasPoema[-1]
  nverso += 1
```

[13] Como todos los poemas empezarán con una primera estrofa, independientemente de que sea la única o de su extensión, escribimos esto también directamente como algo fijo e invariable.

```
  cverso ='{:04.0f}'.format(nverso)
  salida = salida + '               <l
xml:id="P{}V{}">{}</l>\n'.format(idPoema, cverso, linea)
```

Una vez definidas todas las variables que necesitamos, creamos un bucle que recorra todas las líneas una por una, menos la última, que recibe un tratamiento diferente, ya que en nuestro documento todos los versos salvo el último incluyen un salto de línea —«\n»— en la última posición. Como queremos que cada verso anotado contenga la etiqueta de cierre </l> en esa misma línea, y no en la siguiente, nuestro código debe ignorar este último carácter en todas las líneas salvo en la última.

Establecemos, así, el primer caso: si la línea contiene un solo carácter, este será un salto de línea («\n»). Esta línea en blanco representará un cambio de estrofa, por lo que nuestro programa aumentará un número en el contador de estrofas, formateará el número para que sea de cuatro dígitos —añadiendo delante automáticamente los ceros que sean necesarios— y reescribirá la variable «salida» añadiendo el cierre de la estrofa anterior y el inicio de una nueva con su correspondiente identificador. La segunda casuística aplica a las líneas que sí contienen texto. El contador para los versos sigue el mismo proceso que el contador de las estrofas, y el elemento «salida» se reescribe añadiendo en esta ocasión la etiqueta <l>, el identificador, el texto y el cierre </l>. finalmente, se añade de nuevo el salto de línea anteriormente ignorado para que el siguiente verso se escriba en una línea nueva. En último lugar, tratamos el último verso del poema según lo explicado anteriormente.

```
  # el usuario introduce el nombre que quiere para su archivo .tei.xml y
  # se genera
  print("Indica el nombre del archivo TEI transformado que quieres obtener")
  outFile = input("Nombre del archivo (sin extensión): ") + ".tei.xml"


  with open(outFile, mode="w", encoding="utf-8") as poemaAnotado:
    # escribimos todo, incluyendo la cabecera y el cierre de TEI
    poemaAnotado.write(header)
    poemaAnotado.write(head)
    poemaAnotado.write(body)
    poemaAnotado.write(salida)
    poemaAnotado.write(footer)
```

En el último input se da al usuario la opción de elegir qué nombre tendrá el archivo que va a descargar, añadiendo el código de manera automática la extensión del mismo —.tei.xml—. Creamos un documento en el que escribimos todas las variables de texto fijas que definimos al principio y la variable «salida» que contiene la anotación del poema.

Por su parte, en Google Colaboratory tenemos un cuaderno de Jupyter Notebook compuesto por tres celdas. En esta primera, utilizamos la librería «files» para importar y exportar archivos. Guardamos el archivo bajo el nombre variable «archivoPoema».

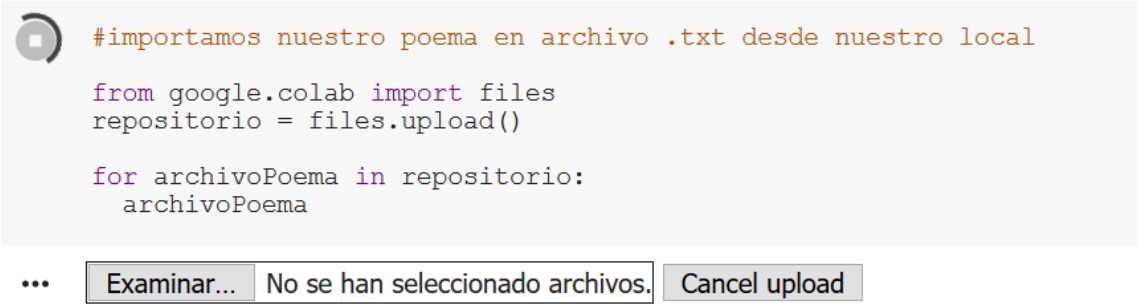

Figura 10. Primera celda del código en Google Colaboratory. Subida del archivo

En la siguiente celda se encuentra el mismo código que explicamos anteriormente, con la salvedad de que no pedimos al usuario que introduzca el nombre del archivo ni añadimos la extensión .txt puesto que ya ha sido almacenado en el programa.

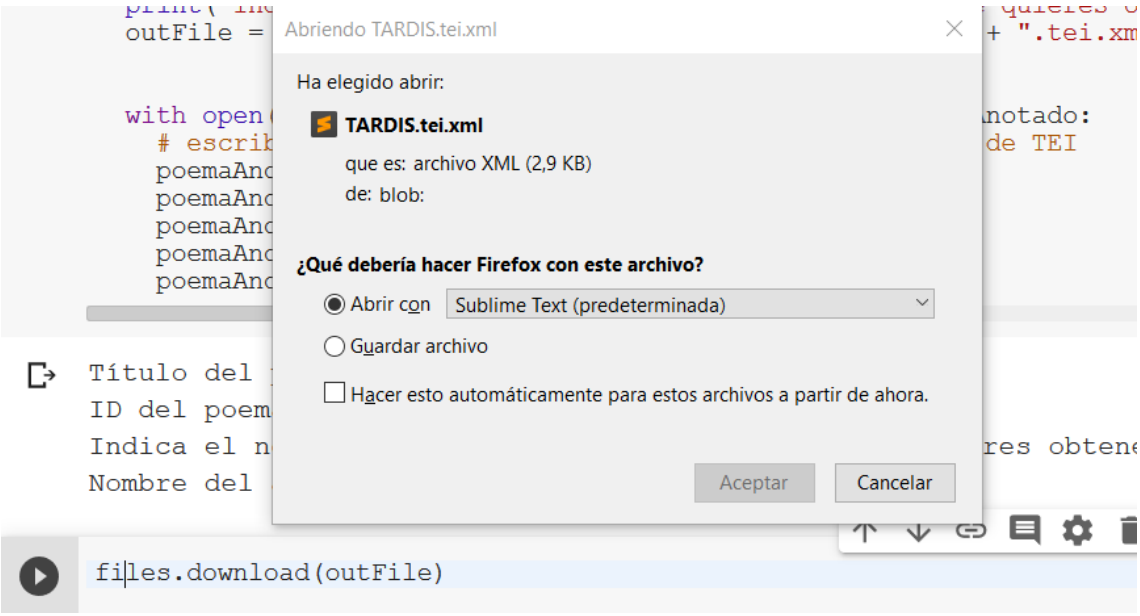

Figura 11. Tercera celda. Descarga del archivo

Para finalizar, en la tercera y última celda se recupera la librería de Google Colaboratory, esta vez para descargar el archivo. Se añade como una facilidad, aunque no sería necesario ya que Google Colaboratory cuenta con una columna a la izquierda donde podemos ver todos los archivos que hemos subido o generado durante la última sesión.

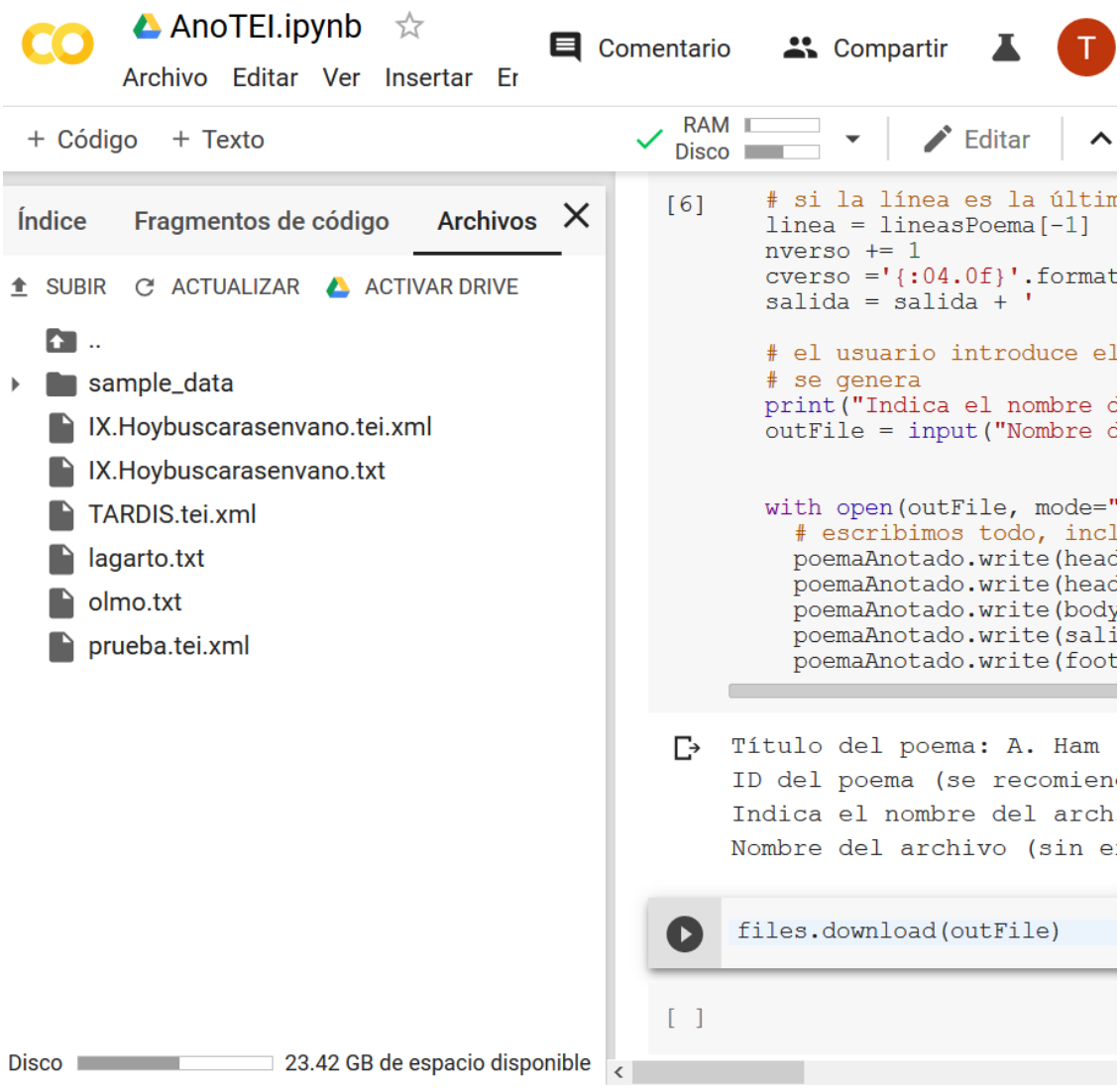

Figura 12. Vista general del código y de la columna donde se ven los archivos de la sesión actual

Como nota final, queremos señalar que en un principio nos planteamos la posibilidad de utilizar la API ElementTree[14] (Lundh, 2017) para realizar el etiquetado, pues esta librería permite crear el árbol de elementos y subelementos del XML de una manera sencilla y mucho más limpia en lo que al código se refiere. Sin embargo, al introducir lo contenido en «salida», realizaba una conversión de los caracteres «<» y «>» en «<» y «>», respectivamente:

[14] https://docs.python.org/2/library/xml.etree.elementtree.html#

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title> </title>
      </titleStmt>
      <publicationStmt> </publicationStmt>
      <sourceDesc> </sourceDesc>
    </fileDesc>
  </teiHeader>
  <body>
    <text>&lt;lg&gt;
           &lt;l&gt;Con diez cañones por banda,&lt;/l&gt;
           &lt;l&gt;viento en popa a toda vela,&lt;/l&gt;
           &lt;l&gt;no corta el mar, sino vuela&lt;/l&gt;
           &lt;l&gt;un velero bergantín;&lt;/l&gt;
        &lt;/lg&gt;
        &lt;lg&gt;
           &lt;l&gt;bajel pirata que llaman,&lt;/l&gt;
           &lt;l&gt;por su bravura, el Temido,&lt;/l&gt;
           &lt;l&gt;en todo mar conocido&lt;/l&gt;
           &lt;l&gt;del uno al otro confí&lt;/l&gt;
</text>
  </body>
</TEI>
```

Figura 13. Resultado inesperado al aplicar ElementTree

```
import xml.etree.ElementTree as ET
from html import unescape

tei = ET.Element('TEI', xmlns='http://www.tei-c.org/ns/1.0')
tei_header = ET.SubElement(tei, 'teiHeader')
filedesc = ET.SubElement(tei_header, 'fileDesc')
file_titlestmt = ET.SubElement(filedesc, 'titleStmt')
titlemain = ET.SubElement(file_titlestmt, 'title')
titlemain.text = ' '
file_publicationstmt = ET.SubElement(filedesc, 'publicationStmt')
file_publicationstmt.text = ' '
file_sourcedesc = ET.SubElement(filedesc, 'sourceDesc')
file_sourcedesc.text = ' '
tei_body = ET.SubElement(tei, 'body')
tei_text = ET.SubElement(tei_body, 'text')
```

Figura 14. Fase anterior del código con el intento de implementación de ElementTree (1).

```
abrel="<l>"
cierral="</l>\n"
salida = "<lg>\n"
cierralg = "        </lg>\n"
with open('pirata.txt', 'r') as f:
  lineas = [linea for linea in f]

for linea in lineas:
  if len(linea)==2:
    salida=salida+"        </lg>\n        <lg>\n"

  else:
    linea = linea [:-2]
    salida=salida+"            "+abrel+linea+cierral

tei_text.text = salida

# print/write the result

indent(tei)
ET.dump(tei)
tree = ET.ElementTree(tei)
tree.write("test1.xml", encoding='utf-8', xml_declaration=True)
```

Figura 15. Fase anterior del código con el intento de implementación de ElementTree (2).

Al introducir, en una etapa posterior del desarrollo del código, el método format(), con el cual insertamos el valor de una variable dentro de una cadena de caracteres, probamos de nuevo la implementación de ElementTree, pero en esta ocasión nos encontramos con otro fallo, ya que Python no nos permitía utilizar el carácter de los dos puntos «:» de este modo, mientras la sintaxis de ElementTree nos obligaba a escribirlo de esta manera (en lugar de, por ejemplo, hacerlo como una cadena de caracteres):

```
poem_lg = ET.SubElement(tei_text, 'lg', xml:id=P)
```

```
  File "<ipython-input-12-64b021c7cac6>", line 24
    poem_lg = ET.SubElement(tei_text, 'lg', xml:id=P)
                                                   ^
SyntaxError: invalid syntax
```

Figura 16. Error de Python por el atributo «xml:id»

Para finalizar, queremos recalcar que para que el programa devuelva el resultado esperado, el archivo de texto plano debe respetar el formato que se ha descrito, con las estrofas separadas únicamente con una línea totalmente en blanco (sin espacios, asterisco o cualquier otro carácter). Es recomendable además que, al guardar el archivo .txt en nuestro ordenador, se especifique la codificación UTF-8, pues esto ahorrará problemas con los caracteres muy comunes en español como pueden ser las vocales con tildes o la letra eñe.

4.3. DEL XML AL HTML: DISEÑO DE LA INTERFAZ Y FUNCIONALIDADES UX/UI

Tras la delimitación del corpus literario, así como del desarrollo del programa de automatización del marcado XML-TEI, procedimos a la elección de las funcionalidades del portal web, atendiendo a los objetivos del proyecto y al grupo de usuarios al que se dirigiría el diseño del mismo. Declaramos, por tanto, las siguientes funcionalidades:

1. Contenedor de intertextos, con información bibliográfica y acceso a la lectura completa del recurso.
2. Contenedor de etiquetas temáticas y/o semánticas acerca del texto a tratar.
3. Marcadores de información métrica. Uno mediante el cual se subraye la rima de cierto color, en cada verso; y otro, mediante el cual se ofrezca un desglose del esquema métrico y del tipo de estrofas que componen cada poema.
4. Integración de un menú que permita regresar a la portada de la edición, ya sea a la muestra de obras base, o a la de intertextos.
5. Enriquecimiento del texto con tooltips emergentes explicativos.
6. Integración de un plugin con sistema de anotación del texto sobre el portal web.
7. Integración de un plugin que permita acceder al portal con perfil de usuario, así como la participación en foros nativos.

El diseño habría de responder, además, a una serie de criterios pertinentes para el desarrollo del proyecto, que se relacionarían de forma directa con la filosofía adoptada por el equipo, en aras de contemplar una posible difusión del prototipo. Además de ofrecer una vista atractiva, de corte minimalista y *responsive*, la interfaz debía resultar accesible, tanto por su carácter inclusivo, como por su naturaleza, que consideraríamos compatible con la definición propia de los sitios web publicados en acceso abierto, u *Open Access*. Por añadidura, contemplamos el prototipo de edición hiperintertextual como un contenido web reutilizable y de libre disposición, así como de código abierto, en lo referente al programa de marcado automático AnoTEI, por lo que lo incluimos en el portal.

4.3.1. Generación de documentos XML y marcado TEI

Seguidamente, el corpus literario es sometido a un proceso de etiquetado o marcado, siguiendo la codificación de los estándares XML de la TEI (*Text Encoding Iniciative*). Este punto del diseño de la interfaz se correspondería con la conformación de la estructura

de la web semántica, esto es, del entramado taxonómico sobre el cual se cimentará la clasificación de los distintos elementos que constituirán nuestra particular visión de los textos en nuestro prototipo. De forma específica, el marcado de los poemas se completa mediante dos fases de etiquetado:

1) un primer estadio, automatizado, en el que el texto plano, con extensión .txt, es traducido a un esquema TEI mediante un programa codificado en Python (§4.2);
2) y un segundo periodo, correspondiente a un proceso de etiquetado manual.

El archivo .xml obtenido a través de la codificación automática, implementada por el programa AnoTEI, muestra, como bien se ha apuntado anteriormente, la cabecera necesaria en todo documento TEI, así como la anotación de los elementos <l>, para cada línea —en este caso, para cada verso— y de <lg> para cada grupo de líneas —esto es, para cada estrofa—. Además, cada uno de estos componentes dispone de un identificador único, lo cual termina por reducir sustancialmente el tiempo dedicado por el usuario en el proceso de marcado de los textos.

A continuación, en la fase de marcado manual, completamos la información contenida en cada uno de los versos, dirigida a cubrir satisfactoriamente las exigencias propias de las funcionalidades que habría de tener, en un futuro, la edición implementada en el portal web. Por un lado, en el interior de cada estrofa, anotadas con la etiqueta <lg>, encontraríamos los atributos @type, @met y @rhyme, que se corresponderían, de forma sucesiva, con el tipo de estrofa, el esquema métrico y el esquema de rima. Asimismo, en cada verso, anotados con la etiqueta <l> en TEI, el elemento <rhyme> señalaría el punto concreto en el que se sitúa la rima.

Por otro lado, marcamos las áreas temáticas o *loci* comunes del corpus delimitado haciendo uso del elemento <seg>[15], con el que pueden identificarse, en cada verso, subcomponentes con contenido propio, y cuya tipología ha de ser etiquetada mediante el atributo @type. En este caso, decidimos adherirnos a la terminología propia de la clasificación temática de artículos en la blogosfera, categorizando a los *loci* comunes como *tags*.

4.3.1.1. Etiquetas personalizadas

Uno de los principales objetivos de nuestro trabajo consiste en la marcación intertextual de los poemas. Como bien se ha explicado anteriormente, entendemos

[15] https://www.tei-c.org/release/doc/tei-p5-doc/en/html/VE.html#VESE

intertexto como la relación entre dos textos, por influencia de uno sobre el otro de una forma más o menos explícita en el propio texto literario (§Estado de la cuestión).

Tras realizar varias búsquedas exhaustivas en TEI, analizando etiquetas que podrían utilizarse para anotar este tipo de relaciones, vimos que ninguna se correspondía lo suficiente con lo teorizado por Genette. Incluso buscando artículos que hablasen sobre cómo referenciar esto en TEI, nos encontramos con muy poca bibliografía, como sucedía en el caso del proyecto Orlando, también referenciado en epígrafes previos, en el que la anotación de fenómenos como el de la intertextualidad se había pensado específicamente para el marcado en SGML, dirigido a formar parte del contenido de una base de datos de acceso cerrado, sujeto a suscripción.

Por este motivo, decidimos proponer, en principio, la creación de dos nuevas etiquetas en nuestro proyecto: <transtextuality> e <intertext>[16], que estarían dispuestas de la siguiente manera en el poema de Antonio Machado que hemos utilizado como ejemplo a lo largo del trabajo:

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
   <teiHeader>
      <fileDesc>
         <titleStmt>
            <title></title>
         </titleStmt>
         <publicationStmt>
            <p></p>
         </publicationStmt>
         <sourceDesc>
            <p></p>
         </sourceDesc>
      </fileDesc>
   </teiHeader>
   <text>
      <body>
         <head>Hoy buscarás en vano</head>
         <lg xml:id="P0009" type="silva arromanzada">
            <lg xml:id="P0009E0001" type="silva arromanzada" met="7, 7"
rhyme="-a">
               <l xml:id="P0009V0001">Hoy buscarás en v<rhyme>ano</rhyme></l>
               <l xml:id="P0009V0002">a tu <seg type="tag">dolor</seg>
consu<rhyme>elo</rhyme>.</l>
            </lg>
```

[16] Decidimos crearlas en inglés para que se adapten al estándar de TEI.

```
            <lg xml:id="P0009E0002" type="silva arromanzada" met="7, 7, 7, 7,
7, 11" rhyme="-a-a-A">
               <l xml:id="P0009V0003">Lleváronse tus h<rhyme>adas</rhyme></l>
               <l xml:id="P0009V0004">el lino de tus <seg
type="tag">su<rhyme>eños</rhyme></seg>.</l>
               <l xml:id="P0009V0005">Está la fuente m<rhyme>uda</rhyme>,</l>
               <l xml:id="P0009V0006">y está marchito el
hu<rhyme>erto</rhyme>.</l>
               <l xml:id="P0009V0007">Hoy sólo quedan
l<rhyme>ágrimas</rhyme></l>
               <l xml:id="P0009V0008">para llorar. No hay que llorar,
¡sil<rhyme>encio</rhyme>!</l>
            </lg>
         </lg>
         <transtextuality>
         <intertext xml:id="P0009INT0001">
            <head>Como una inundación</head>
            <p>Poema de Félix Grande. Nivel intertextual.</p>
            <bibl type="book" subtype="anthology">
               <author>Félix Grande</author>
               <title>Biografía: poesía completa (1958-1984)</title>
               <publisher>Anthropos</publisher>
               <date>1989</date>
            </bibl>
         </intertext>
         <intertext xml:id="P0009INT0002">
            <head>El lino de los sueños</head>
            <p>Título del primer libro de Alonso Quesada. Nivel
intertextual.</p>
            <bibl type="book" subtype="poetry">
               <author>Alonso Quesada</author>
               <title>El lino de los sueños</title>
               <date>1915</date>
            </bibl>
         </intertext>
         </transtextuality>
      </body>
   </text>
</TEI>
```

Siguiendo el esquema terminológico de Genette, dentro de la etiqueta <transtextuality> se englobarían el resto de elementos correspondientes a los tipos de transtextualidad existentes, es decir, dentro de él podríamos crear las etiquetas

<intertext>, <architext>, etc[17]. Nos referiremos, no obstante, únicamente a la primera, por ser la más común y la que consideramos necesaria para nuestro trabajo.

Así, la etiqueta <intertext> que proponemos incluiría el elemento de TEI <bibl>, utilizado para indicar datos bibliográficos como autor, título, fecha, editorial, etc. añadimos, además, para el HTML final, un elemento <head> con el título del poema y un elemento con una pequeña descripción para ser visualizada en las tarjetas.

Para explicar mejor la naturaleza de estas etiquetas dentro del propio estándar de TEI, la visualización que tendrían en la versión en línea de sus directrices sería algo parecido a lo siguiente:

<transtextuality>

<transtextuality> (transtextualidad) indica la relación entre dos textos	
Módulo	core — Elements Available in All TEI Documents
Atributos	att.global (@xml:id, @n, @xml:lang, @xml:base, @xml:space) (att.global.rendition (@rend, @style, @rendition)) (att.global.linking (@corresp, @synch, @sameAs, @copyOf, @next, @prev, @exclude, @select)) (att.global.analytic (@ana)) (att.global.facs (@facs)) (att.global.change (@change)) (att.global.responsibility (@cert, @resp)) (att.global.source (@source)) att.typed (@type, @subtype) att.placement (@place) att.written (@hand)
Miembro de	model.transtextuality
Contenido en	**core:** divGen lg list listBibl **drama**: castGroup castList epilogue performance prologue set spGrp **figures**: figure table **msdescription**: msDesc msFrag msPart **textstructure**: back body div div1 div2 div3 div4 div5 div6 div7 front group postscript
Puede contener	**core**: architext front hipertext intertext metatext
Ejemplo	**<transtextuality>** **<intertext xml:id="P0001INT0001" type="intertext">** **<head>**En nuestras almas todo**</head>** ** **Poema de Antonio Machado. Nivel intertextual.** ** **<bibl type="book" subtype="poetry">** **<author>**Antonio Machado**</author>** **<title>**Soledades (Poesías)**</title>** **<publisher>**Imp. de A. Álvarez**</publisher>** **<date>**1903**</date>** **</bibl>** **</intertext>** **</transtextuality>**

Figura 17. Especificaciones de la etiqueta <transtextuality> a modo del modelo TEI

[17] A excepción de la etiqueta <paratext>, puesto que el modelo TEI ya cuenta con una etiqueta <front> que se utiliza precisamente para este fin.

<intertext>

<intertext> (intertexto) indica una relación entre dos textos de cualquier tipo	
Módulo	core — Elements Available in All TEI Documents
Atributos	att.global (@xml:id, @n, @xml:lang, @xml:base, @xml:space) (att.global.rendition (@rend, @style, @rendition)) (att.global.linking (@corresp, @synch, @sameAs, @copyOf, @next, @prev, @exclude, @select)) (att.global.analytic (@ana)) (att.global.facs (@facs)) (att.global.change (@change)) (att.global.responsibility (@cert, @resp)) (att.global.source (@source)) att.typed (@type, @subtype)
Miembro de	model.transtextuality
Contenido en	**core:** transtextuality
Puede contener	**core**: architext bibl biblStruct front head hipertext intertext metatext l lg p q quote ref reg title
Ejemplo	**<intertext xml:id="P0001INT0001" type="intertext">** **<head>**En nuestras almas todo**</head>** ** **Poema de Antonio Machado. Nivel intertextual.** ** **<bibl type="book" subtype="poetry">** **<author>**Antonio Machado**</author>** **<title>**Soledades (Poesías)**</title>** **<publisher>**Imp. de A. Álvarez**</publisher>** **<date>**1903**</date>** **</bibl>** **</intertext>**

Figura 18. Especificaciones de la etiqueta <intertext> a modo del modelo TEI

4.3.2. Marcado con estilo: la hoja de transformaciones XSLT

Atendiendo al diseño pactado para las páginas que, en el portal web, alojarían cada uno de los poemas y de sus intertextos, así como sus funcionalidades, desechamos la generación de documentos HTML de forma original, decantándonos por la maquetación de una hoja de transformaciones XSLT, con el fin de automatizar lo máximo posible el proceso de implementación de la obra en el portal web.

XSLT es un lenguaje de programación declarativo que permite generar documentos a partir de otro documento inicial, codificado en XML. Mediante el procesamiento de la hoja de transformaciones XSLT, hemos podido desarrollar una plantilla HTML perfectamente visible en cualquier navegador actual, haciendo uso, además, de las atractivas bibliotecas CSS y JS incluidas en el conocido *framework* de Twitter: Bootstrap, en su última versión.

```
<xsl:template match="/">
   <html lang="es">
      <head>
         <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

```
         <meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1"/>
         <!-- Bootstrap stylesheet -->
           <link rel="stylesheet" href="../bootstrap/css/bootstrap.css"/>
           <!-- Our stylesheet -->
           <link rel="stylesheet" type="text/css" href="../css/style.css"/>
           <link rel="stylesheet" type="text/css"
href="../css/xslstyle.css"/>
           <!-- jQuery library -->
           <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></scri
pt>
           <!-- Bootstrap JS-->
           <script src="../bootstrap/js/bootstrap.min.js"></script>
           <!--Our JS-->
           <script src="code.js"></script>
           <title>
              Transformado automáticamente de XML TEI anotado a HTML
           </title>
         <!-- <title><xsl:value-of
select="tei:TEI/tei:text/tei:body/tei:head"/></title> -->
      </head>
```

Figura 19. Cabecera del archivo .xsl y cabecera del HTML a generar

```
<xsl:template match="tei:TEI/tei:text/tei:body/tei:head">
   <h1 class="display-4 separador-titulo">
      <xsl:apply-templates/>
   </h1>
</xsl:template>
```

Figura 20. Fragmento del esquema XSLT con aplicación de *templates* (1)

```
<xsl:template match="*/tei:lg/tei:lg">
   <p class="estrofa" hidden="true">
      Estrofa | Métrica | Rima: <xsl:value-of select="@type"/> | <xsl:value-
of select="@met"/> | <xsl:value-of select="@rhyme"/>
   </p>
   <p>
      <xsl:apply-templates/>
   </p>
</xsl:template>
```

Figura 21. Fragmento del esquema XSLT con aplicación de *templates* (2)

Solo Internet Explorer y Edge, los navegadores de Microsoft, soportan la transformación automática de XML a HTML mediante la hoja de estilos XSLT. Por motivos de seguridad, tanto Google Chrome como Mozilla Firefox, dos de los navegadores más populares, no ofrecen la posibilidad de visualizar una web de forma

directa a través de una hoja de transformaciones XSLT. Por ello, para la correcta visualización del portal, realizamos una conversión de todos los documentos XML a formato HTML automáticamente, desde consola, utilizando la herramienta xsltproc[18], que permite aplicar una hoja de transformaciones XSLT a un XML para más tarde volcar el resultado en un nuevo archivo.

La generación de este archivo final pasa por una primera lectura secuencial del documento XML, en el que, por cada una de las etiquetas y del contenido de estas, el procesador XSLT busca instrucciones dedicadas a su conversión. En el caso del poema utilizado como ejemplo a lo largo de nuestro trabajo, al alcanzar la etiqueta <head>, el procesador XSLT aplica las instrucciones contenidas en la siguiente plantilla: <xsl:template match="tei:TEI/tei:text/tei:body/tei:head">, correspondiente al marcado del título del poema. En el interior de dicha plantilla se encontraría alojada la etiqueta <h1>, utilizada en HTML para definir el encabezado principal. Con ella, se pretende marcar la manera en la que se visualizará el XML en el navegador, tras el proceso de transformación de este primer documento en HTML.

```
<xsl:template match="tei:TEI/tei:text/tei:body/tei:head">
   <h1 class="display-4 separador-titulo">
      <xsl:apply-templates/>
   </h1>
</xsl:template>
```

Por otro lado, realizando pruebas en el CMS Wordpress con idénticos fines a los anteriormente descritos (esto es, procesar satisfactoriamente la hoja de estilos XSLT), implementamos un código Javascript a nivel cliente que permitiera, en cada una de las páginas del portal, llevar a cabo las transformaciones pertinentes de XML a HTML, pasando por el XSLT. Por dicha razón, disponemos de tres documentos distintos para la conformación de la vista de los poemas en el navegador:

1. un archivo marcado en lenguaje XML, siguiendo los estándares de TEI, que hemos guardado bajo el membrete poema00mk.xml;
2. una hoja de transformaciones XSLT, común a todos los poemas, a la que hemos llamado transformacion.xsl;
3. una *template* PHP, propia de cada texto, almacenada con el nombre de poema00tp.php, que contiene la base web de cada página (barra de navegación y menú), así como el código Javascript anteriormente anunciado.

```
<?php
```

[18] http://xmlsoft.org/XSLT/xsltproc2.html

```
/*
Template Name: Poema09Estilo
*/
?>
<script>
function loadXMLDoc(filename)
{
if (window.ActiveXObject)
  {
  xhttp = new ActiveXObject("Msxml2.XMLHTTP");
  }
else
  {
  xhttp = new XMLHttpRequest();
  }
xhttp.open("GET", filename, false);
try {xhttp.responseType = "msxml-document"} catch(err) {} // Helping IE11
xhttp.send("");
return xhttp.responseXML;
}

function displayResult()
{
xml = loadXMLDoc("../wp-content/themes/TFM-LLDD/poema09mk.xml");
xsl = loadXMLDoc("../wp-content/themes/TFM-LLDD/transformacion.xsl");
// code for IE
if (window.ActiveXObject || xhttp.responseType == "msxml-document")
  {
  ex = xml.transformNode(xsl);
  document.getElementById("example").innerHTML = ex;
  }
// code for Chrome, Firefox, Opera, etc.
else if (document.implementation && document.implementation.createDocument)
  {
  xsltProcessor = new XSLTProcessor();
  xsltProcessor.importStylesheet(xsl);
  resultDocument = xsltProcessor.transformToFragment(xml, document);
  document.getElementById("example").appendChild(resultDocument);
  }
}
</script>
```

Desde un punto de vista técnico, y a pesar de que este método de implementación por *templates* ligadas a cada uno de los poemas nos ofrecía el resultado que esperábamos, no suponía una forma eficaz de integración del contenido en el portal web. Lo ideal, en cambio, habría sido el disponer de una única plantilla PHP, que pudiera ser procesada por

cada uno de los poemas, en su conversión o transformación a HTML. Sin embargo, no poseíamos los conocimientos específicos para realizar esta tarea, por lo que recurrimos a la primera opción, con tal de conseguir la muestra que deseábamos.

Figura 22. Aplicación de plantilla o *template* en página de WordPress

Figura 23. Multitud de plantillas: una para cada poema

4.3.3. Implementación del portal web y funcionalidades

Haciendo uso de la CSS predefinida por Bootstrap, así como a través de una hoja de estilos propia (style.css), terminamos por maquetar el portal web, partiendo, por un lado, de tres páginas HTML, y, por otro, de los HTML generados como resultado de la transformación XSLT de los documentos iniciales, marcados con XML. Así, el sitio está

formado por tres páginas distintas: *Inicio*, *Edición* y *Sobre nosotras*. Todas ellas se organizan de forma similar, mostrando, en orden, los siguientes elementos:

a) Una barra de navegación interactiva, con el logo de la Universidad Complutense de Madrid, y marca de posicionamiento para el usuario;
b) un título sobre una imagen de presentación, diseñada en Photoshop, y que guarda relación con el contenido (en este caso, se trata de ilustraciones de Antonio Machado);
c) el contenido, dividido en *tarjetas* o *cards* de Bootstrap, en su mayor parte;
d) y un *footer*, en el cual aparece una inscripción, indicando el nombre del equipo implicado en el desarrollo del portal web, así como una imagen del logo del Máster en Letras Digitales.

Además, todas las páginas han sido maquetadas bajo un diseño *responsive*, aprovechando las facilidades de Bootstrap para tales efectos, de manera que puedan adaptar su vista a dispositivos de tamaños distintos a los de un ordenador, como puede ser el caso de las *tablets* o de los *smartphones*. En las siguientes figuras puede apreciarse la diferencia entre ambos modelos de vista desde la página de Inicio:

Figura 24. Vista en ordenador

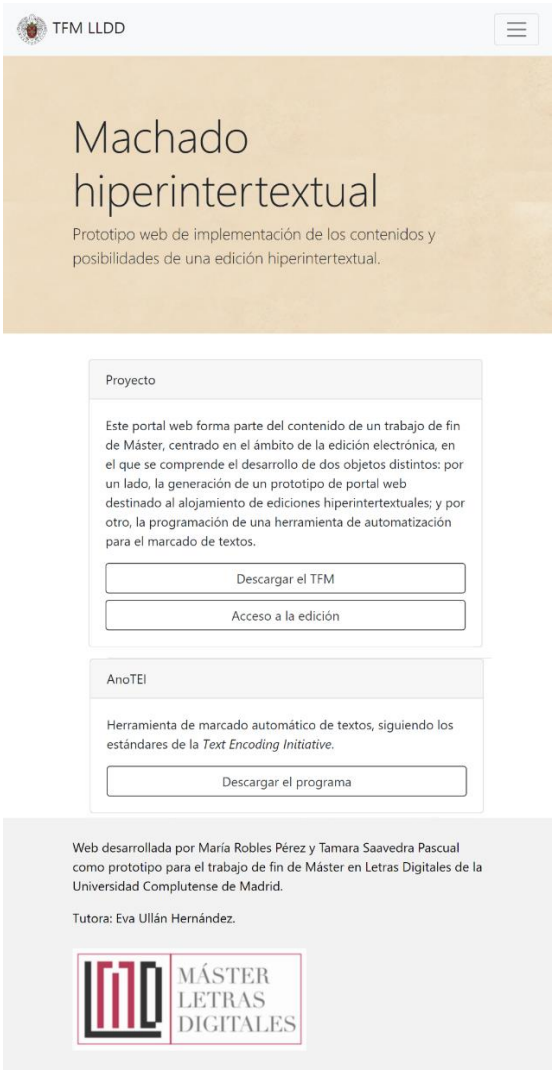

Figura 25. Vista en *smartphone*

A continuación, mostramos la vista de la página de Edición, que constituiría el núcleo del trabajo. En ella, el usuario dispondría de un menú general, situado a la izquierda del contenedor del cuerpo del documento HTML, a través del cual podría acceder a las obras principales del corpus, o a los intertextos que los acompañan, de forma separada. A la derecha, por tanto, el usuario podría acceder a la lectura de las obras del corpus, cuya disposición se ha llevado a cabo mediante *tarjetas* o *cards*, propias del diseño de Bootstrap, a lo largo y ancho del resto de la página. Adjuntamos, además, una captura del aspecto de la página del noveno poema de *Galerías*, al cual se accede a través de la referida página de Edición. En esta última, se muestra el poema del corpus elegido por el usuario, el intertexto relacionado en la parte inferior de la página y un pequeño contenedor, a la derecha, en el que aparecen dos botones: uno para marcar la rima del poema, y otro para desvelar información métrica sobre el mismo.

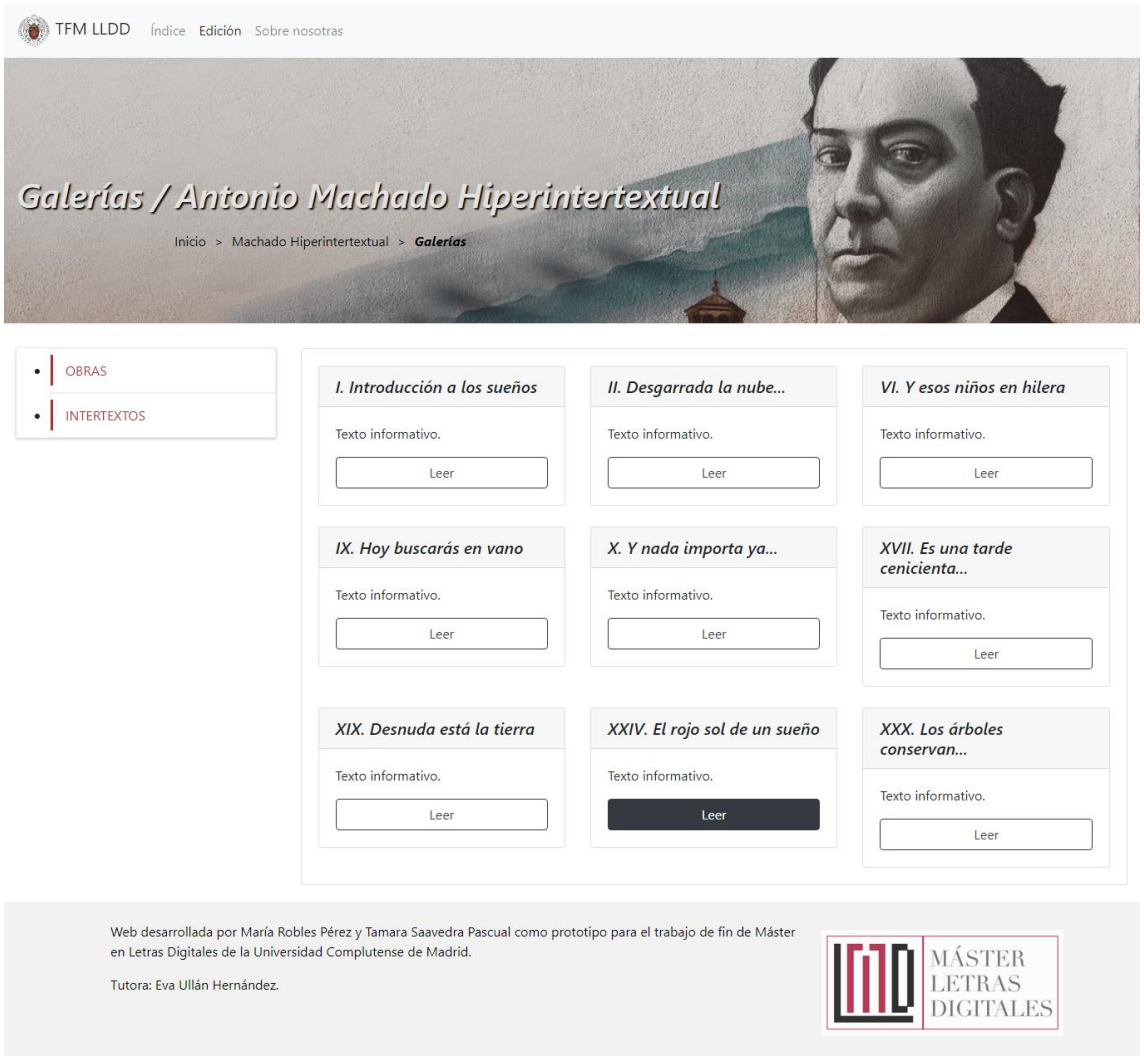

Figura 26. Vista de la página Edición

Figura 27. Vista del poema IX

Sobre el portal web, se aplicaron tres funcionalidades distintas: dos de ellas, anteriormente referidas (marcado de rima y muestra de información métrica), en la página

de vista de cada uno de los poemas; y otra, aparte, que posibilita la anotación del contenido publicado en el portal web mediante un *script* de desarrollo externo, ajeno a nuestro trabajo, pero que comulga perfectamente con el espíritu del mismo.

El pintador o marcador de rima colorea la parte final de cada verso, siguiendo las normas de la métrica en español. Su implementación ha sido posible mediante el desarrollo de una función específica a tal efecto, en un archivo Javascript que, a su vez, se ligó a la hoja de transformaciones XSLT. En los documentos iniciales, marcados en XML, colocamos una etiqueta <rhyme> en cada uno de los versos, con tal de anotar la rima debidamente en todos los poemas.

```
<lg xml:id="P0009E0001" type="silva arromanzada" met="7, 7" rhyme="-a">
   <l xml:id="P0009V0001">Hoy buscarás en v<rhyme>ano</rhyme></l>
   <l xml:id="P0009V0002">a tu <seg type="tag">dolor</seg>
consu<rhyme>elo</rhyme>.</l>
</lg>
```

A raíz de ella, en la hoja de transformaciones XSLT se incluyó una *template* o plantilla que, mediante XPath, seleccionaba por completo el conjunto de nodos que, en el marcado XML, contuvieran la susodicha etiqueta <rhyme>. Más tarde, en la conversión del archivo inicial a HTML, el botón «Rima» llama a la función changecolor(), que es la siguiente:

```
function changecolor() {
   var x = document.querySelectorAll(".rhyme");

   var i;
   for (i = 0; i < x.length; i++) {
      x[i].classList.toggle("red");
   }
}
```

En primer lugar, definimos una variable «x» con la que seleccionamos todos los elementos que contengan la clase «rhyme», puesto que, aunque eran etiquetas en el XML, el XSLT tiene una plantilla que convierte todas ellas a un elemento de HTML «span» con una clase «rhyme»:

```
<xsl:template match="*/tei:rhyme">
   <span class="rhyme">
      <xsl:apply-templates/>
   </span>
</xsl:template>
```

Todos los elementos seleccionados se guardan en una lista, y podríamos acceder a cada uno de ellos seleccionando su posición en la misma —[0] para el primer verso, [1]

para el segundo, etc.—. No obstante, queremos que la función se aplique a todos los versos, no solo de este poema sino de todos los que incluyamos independientemente de su extensión —y, por tanto, de la longitud de la lista que genera el código—. Por ello, creamos un bucle que cuente todos los y que almacene el resultado final en la variable «i». Finalmente, buscamos en nuestro CSS la clase que contiene el formato que queremos aplicar a nuestros versos, en este caso la que hemos denominado «red», y especificamos que nuestro botón sea «toggle» para que se pueda alternar entre aplicar la función y revertir este cambio volviendo al estado original al pulsarlo de nuevo.

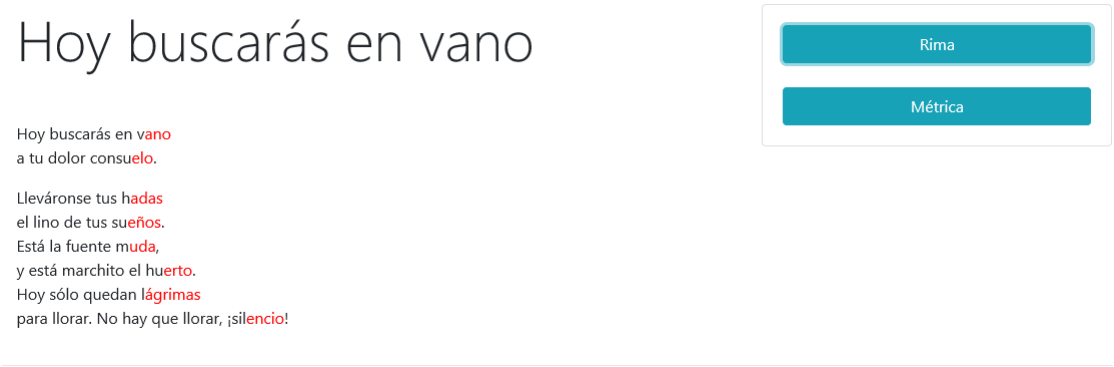

Figura 28. Resultado al pulsar el botón Rima

De igual modo, se llevó a cabo la implementación de la muestra de información métrica sobre el poema de muestra. Los códigos utilizados son idénticos a los que suponen el buen funcionamiento del pintador de rima, aunque la etiqueta XML de la que partimos en este punto no se categoriza como elemento, sino como atributo de <lg>: @type, puesto que marca el tipo de estrofa utilizado en la confección del poema visualizado. En este caso, el valor informativo sobre la métrica de la obra se encierra en una marca de párrafo <p>, a la cual se le ha agregado un estilo mediante el cual se pinta de azul el contenido del mismo, con tal de que pudiéramos destacarlo sobre el resto de componentes de la página.

```
function showmetrica() {
   var x = document.querySelectorAll(".metrica");

   var i;
   for (i = 0; i < x.length; i++) {
      x[i].classList.toggle("visible");
   }
}
```

El botón de métrica llama a la función showmetrica(), cuyo funcionamiento es básicamente el mismo que el del botón anterior. En este caso, la información contenida

por la etiqueta «métrica» incorpora en el XSLT una clase de CSS que la obliga a permanecer oculta en la página. Al pulsar el botón, alternamos esa clase con «visible», también definida en el mismo CSS y que realiza la función exactamente opuesta. Este botón es también «toggle», por lo que revierte los cambios al pulsar de nuevo.

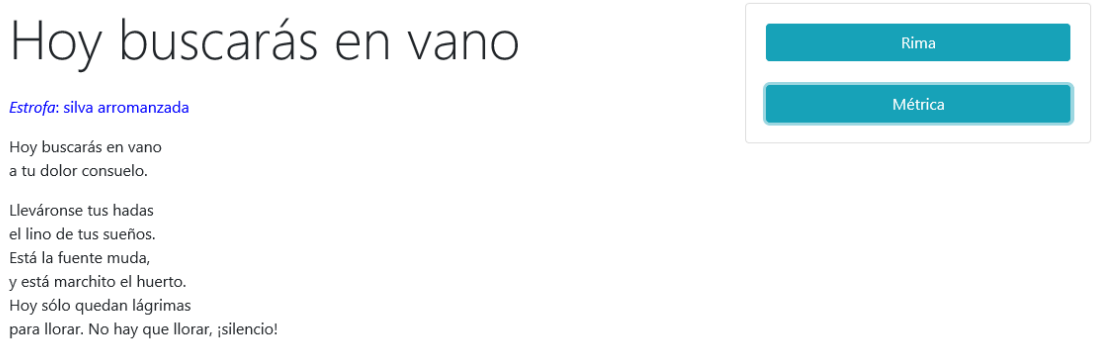

Figura 29. Resultado al pulsar el botón Métrica

Por último, y como se ha mencionado previamente, agregamos un script externo, Hypothesis[19], un software de acceso libre y abierto que permite anotar y subrayar el contenido de cualquier sitio web. Para almacenar esta información, el usuario ha de crear un perfil en el que alojarla y mediante el cual poder compartirla con otros usuarios. La aplicación solo está disponible en inglés, se basa en los estándares de anotación para documentos digitales del Grupo de Anotación de la W3C[20] y se dirige a un amplio rango de potenciales usuarios, como desarrolladores, instituciones académicas o investigadores. La instalación de Hypothesis en web requiere de la inclusión de una línea de código en las páginas HTML donde se desea que sus funciones resulten efectivas:

```
<script src="https://hypothes.is/embed.js" async></script>
```

Consideramos que la aplicación resulta de gran utilidad a nivel usuario, puesto que es intuitiva y sumamente sencilla, aparte de minimalista en su diseño, el cual se integra perfectamente en cualquier tipo de portal web. Además, refleja de forma adecuada el espíritu de parte de nuestros objetivos, en los que planteábamos la inclusión de algún tipo de funcionalidad en la edición que permitiera la anotación o subrayado de parte de su contenido.

[19] https://web.hypothes.is/about/
[20] https://www.w3.org/annotation/

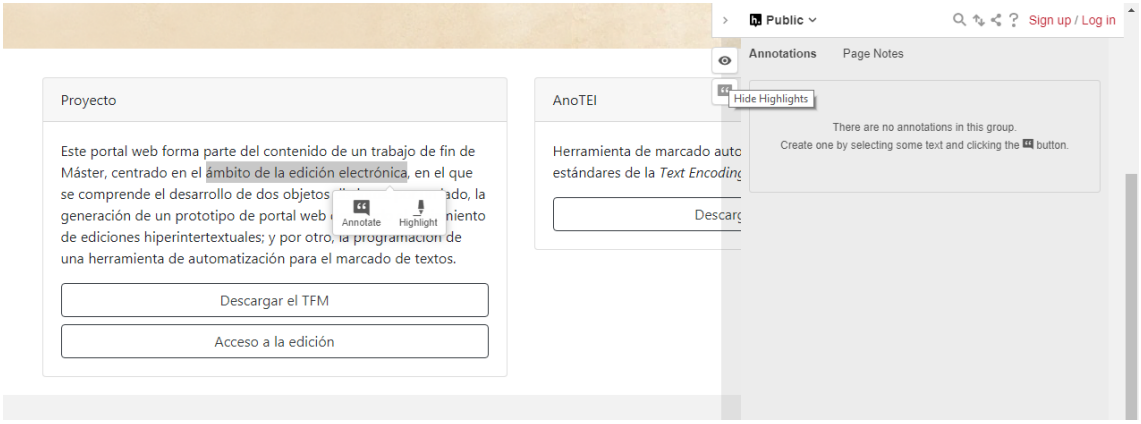

Figura 30. Vista de Hypothesis

5. CONCLUSIONES

Consideramos que en este trabajo de fin de Máster hemos llegado a cumplir prácticamente todos los objetivos que mencionábamos al principio (§3), y que este esbozo de prototipo que hemos dibujado podría seguir creciendo, por lo que analizaremos si los resultados se corresponden con las metas planteadas y propondremos, además, unas líneas de futuro por las cuales se podría seguir trabajando en este proyecto.

5.1. ANÁLISIS DE LOS RESULTADOS Y CUMPLIMIENTO DE OBJETIVOS

En cuanto a los objetivos que nos propusimos, hemos conseguido generar el prototipo de portal web con una interfaz sencilla para alojar ediciones hiperintertextuales en dos soportes diferentes: con el CMS WordPress, por un lado, y con HTML, por el otro, mostrando de esta manera una primera versión del proyecto que tenemos en mente. Sin embargo, no hemos podido añadir en esta primera fase todas las funcionalidades que hubiésemos deseado debido a los problemas que planteó el uso obligatorio de PHP en Wordpress combinado con las hojas XSLT, como los *plugins* que mostrarían la vertiente colaborativa del portal, añadiendo foros, anotaciones o comentarios.

Por otro lado, consideramos teorizada y documentada por completo la implementación de las nuevas etiquetas <transtextuality> e <intertext> para TEI, reinterpretando de este modo el concepto de intertextualidad con las relaciones hipertextuales. Conformamos con ello, además, un nuevo tipo de edición digital en la que los intertextos cobran fuerza, respondiendo así al creciente interés que suscita el estudio de estas relaciones entre textos y ofreciendo la posibilidad de ver *in situ* las obras a las que se refieren, lo cual confiere a la web una navegabilidad mayor de lo habitual, pues las ediciones electrónicas tienden a responder a temáticas más acotadas.

Hemos integrado, asimismo, algunas funcionalidades de marcado métrico como son el pintado de las rimas o la visibilidad de la información métrica del poema, anotadas en TEI pero ocultas en el portal hasta que el usuario decide mostrarlas de manera proactiva, dotándole con diferentes módulos de control sobre la visualización.

También se ha conseguido dar un paso importante para la automatización de las anotaciones, tanto mediante la aplicación de las hojas de estilo XSLT para generar HTML a partir de XML, como con la creación desde cero del programa AnoTEI, cuyas funcionalidades son aún limitadas pero consideramos que suple la carencia que presenta

la mayoría de los programas ya existentes para la anotación TEI respecto a la marcación de obras poéticas.

Por último, mantenemos la ideología de que la plataforma, si llegase en algún momento al uso público, sea accesible y de código abierto, manteniendo la licencia copyleft GNU GPL, o en su defecto la especificación de la licencia de Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional, tanto para la web como para específicamente el programa AnoTEI.

5.2. PERSPECTIVAS DE FUTURO

A lo largo de este trabajo hemos presentado el desarrollo de un prototipo que constituiría la fase inicial de un proyecto mayor. Asentadas las bases de esta primera versión, definimos a continuación las mejoras y ampliaciones que podrían realizarse con el fin de que nuestro portal fuese mucho más completo.

Así, en cuanto al programa AnoTEI, nos gustaría implementarlo en una pestaña dentro de la propia página web, de manera que cualquier usuario pudiese transformar sus documentos en la misma sin necesidad de instalar una consola o de tener que entrar en otra web diferente —Google Colaboratory— para ejecutarlo, como ocurre actualmente. Para ello, sería necesaria la instalación de la web en un servidor, donde se realizaría la ejecución del programa, y el resultado sería enviado al cliente, pudiendo mostrarse en el propio navegador y ofreciendo la opción de descargar un documento nuevo.

También se podrían ampliar las anotaciones automáticas añadiendo más campos a rellenar por el usuario, tales como:

- especificación de otros esquemas además del de TEI;
- valores de los elementos de la cabecera como son el <titleStmt>, <publicationStmt> o <sourceDesc>;
- introducción de palabras clave a las que queramos añadir un etiquetado para tratarlas de manera especial, como sería el caso de los *tags*;
- un desplegable con una lista completa de tipos de estrofas (cuarteto, silva, romance, etc.) para rellenar el atributo *type* de las etiquetas <lg>;
- integración de un contador de sílabas;
- creación de programas similares para la anotación de textos que se correspondan con otros módulos de TEI, como *corpus*, *dictionaries* o *drama*, dando opción al usuario de que elija de qué tipo es el que quiere anotar para que la respuesta sea lo más adecuada posible;

- etc.

Algunas de estas propuestas podríamos haberlas implementado en esta primera versión, al igual que hicimos con el título del poema, pero al ejecutar el programa en entornos que solicitan uno por uno cada campo a añadir, consideramos que ralentizaba de manera considerable el proceso. Es por esto que decidimos dejarlo en algo básico que automatizase la parte común a todos los poemas sin pedir demasiados *inputs*, pues lo ideal sería que el usuario pudiese en una misma pantalla rellenar los campos que le interesasen y dejar en blanco los que no.

En definitiva, existen muchas posibilidades de ampliación para las futuras versiones de AnoTEI, cuyo objetivo principal es mejorar las herramientas de anotación automática ya existentes, incorporando lo que las que mencionábamos no ofrecen a día de hoy. Cabe añadir que la creación de este programa surgió de nuestra necesidad como usuarias de una herramienta que realizase estos procesos, y por ello consideramos que el incremento de funcionalidades de la misma debería hacerse, por un lado, mediante un análisis previo de trabajos que ya se han realizado con anotación TEI, evaluando cuáles son los elementos más utilizados por sus anotadores; y por otro, realizando encuestas a quienes se encuentren actualmente realizando procesos de digitalización con este lenguaje de marcado para ver cuáles son las principales demandas. a fin de establecer un orden de prioridad que se traduzca en un plan de trabajo para la implementación gradual de nuevas funcionalidades.

En otro orden de cosas, una de las mayores ventajas del marcado TEI es que cuenta con numerosas etiquetas y tiene en cuenta diversos tipos de texto, contando con módulos propios para cada uno de ellos, por lo que en general es bastante completo[21]. Sin embargo, esta libertad que ofrece puede tornarse un inconveniente en cuanto a la uniformidad de los textos y las anotaciones, ya que cada proyecto tendrá unas necesidades específicas que afectarán a los elementos seleccionados para la anotación. Además, la existencia de etiquetas muy generales como pueden ser o <label> también juega un papel en contra de la universalización de las anotaciones. Todo esto hace más complicada la tarea de crear una sola interfaz que satisfaga las necesidades de todos los potenciales usuarios.

Queremos señalar que la adición de nuevas etiquetas para el modelo de TEI haría necesaria la especificación de un esquema que contuviera estas nuevas etiquetas para la correcta validación de los documentos XML generados, por lo que este sería uno de los

[21] Salvo en casos específicos y muy puntuales, como hemos visto con el CBML o en este mismo trabajo para el tratamiento de los intertextos.

primeros pasos a dar a continuación, aunque no fuese uno de los objetivos de este trabajo. Existen diferentes tipos de esquemas para validar los documentos XML, como DTD o XML Schema, aunque el más extendido para el marcado TEI es probablemente el escrito con el lenguaje Relax NG[22], con el cual se crean documentos ODD ('*One Document Does it all*'). Algunas herramientas, como ROMA[23], automatizan el proceso de generar estos esquemas de validación.

En cuanto al portal web, nuestro prototipo ha sido creado seleccionando poemas de las *Galerías* de Antonio Machado por las razones anteriormente mencionadas, pero esto no quiere decir que el portal esté restringido a un solo autor o a una sola obra. En un futuro, nuestro deseo sería incluir el mayor número de poemas posibles, de diferentes autores, épocas e incluso idiomas, pues cuanto más completo sea el portal en cuanto a contenido, mayores serán las posibilidades de interrelaciones entre los poemas, bien sean temáticas, intertextuales, cronológicas, por autoría, etc. Apostamos, además, por la implementación de una herramienta de anotación como Hypothesis pero interna a la propia página, y de introducir la gestión de perfiles de usuario, de manera que en cada poema, además del texto y de sus relaciones con otros textos, hubiese un apartado donde los miembros pudiesen incluir anotaciones públicas referentes a la interpretación o el contexto del poema, o bien anotaciones de uso privado solo visibles por el autor para su propio estudio. La plataforma sería, en cierto modo, parecida a la página Genius[24], dedicada a la anotación de letras de canciones, pero para un entorno literario.

[22] https://relaxng.org/

[23] https://roma2.tei-c.org/, que a día de hoy tiene una versión beta con una nueva interfaz disponible en https://romabeta.tei-c.org/ [Fecha de última consulta: 13/09/2019].

[24] https://genius.com/

6. BIBLIOGRAFÍA

FERRER VALLS, T. (DIR.), ET AL., DE ZAYAS, M. (2015). *La traición en la amistad*. Valencia: Grupo Te@doc (Universidad de València). https://dicat.uv.es/te@doc/edicion/TraicionenAmistad_Zayas.html

GARRIDO TEIXEIRA, P. (2017). TeogNet: un portal web dedicado al poeta griego Teognis de Megara, Madrid: e-prints. Consultado en https://eprints.ucm.es/47143/1/TFM%20corregido%20Patricia%20Garrido.pdf [fecha de última visita: 14/09/2019]

GENETTE, G. (1989). *Palimpsestos. La literatura en segundo grado.* Madrid: Taurus, Alfaguara.

GOOGLE (2017). Google Colaboratory. https://colab.research.google.com

LOTMAN, I. M. (1996). *La semiosfera I. Semiótica de la cultura y del texto.* Madrid: Cátedra.

LOZANO, J.; PEÑA-MARÍN, C. Y ABRIL, G. (1982). *Análisis del discurso. Hacia una semiótica de la interacción textual.* Madrid: Cátedra.

LUNDH, F. (2017). Module xml.etree.ElementTree, Python Software Foundation. https://docs.python.org/2/library/xml.etree.elementtree.html#module-xml.etree.ElementTree

MACHADO, A. (2010). *Poesías completas*. Madrid: Austral.

MARTÍNEZ FERNÁNDEZ, J. E. (2001). *La intertextualidad literaria*, Madrid: Cátedra.

MENDOZA FILLOLA, A. (2001). *El intertexto lector.* Barcelona: Universidad de Barcelona. http://www.cervantesvirtual.com/obra-visor/el-intertexto-lector-0/html/01e1dd60-82b2-11df-acc7-002185ce6064_2.html#I_0_

POZUELO YVANCOS, J. M. (1988). *Teoría del lenguaje literario.* Madrid: Cátedra.

PRESOTTO, M. (DIR.) ET AL., DE VEGA, L. (2015). *La dama boba: edición crítica y archivo digital*. Barcelona: Prolope; Bologna: Alma Mater Studiorum - Università di Bologna, CRR-MM. http://damaboba.unibo.it/

SEGRE, C. (1985). *Principios de análisis del texto literario.* Barcelona: Crítica.

TEXT ENCODING INITIATIVE CONSORTIUM, RAHTZ, S. (2011). OxGarage Conversion. https://oxgarage2.tei-c.org/; https://github.com/TEIC/oxgarage/

TEXT ENCODING INITIATIVE CONSORTIUM (2019). *TEI P5: Guidelines for Electronic TextEncoding and Interchange*. https://tei-c.org/guidelines/p5/

WALSH, JOHN A. (2002). *Comic Book Markup Language*, Indiana: Digital Culture Lab, School of Library and Information Science, Indiana University. http://dcl.slis.indiana.edu/cbml/

WALSH, JOHN A. (2012). «Comic Book Markup Language: An Introduction and Rationale», Boston: *Digital Humanities Quarterly (DHQ), 6* (1). Recuperado de http://www.digitalhumanities.org/dhq/vol/6/1/000117/000117.html